

July 2021

Knowledge Extraction and Inference Based on Visual Understanding of Cooking Contents

Ahmad Babaeian Babaeian Jelodar
University of South Florida

Follow this and additional works at: <https://digitalcommons.usf.edu/etd>



Part of the [Computer Sciences Commons](#)

Scholar Commons Citation

Babaeian Jelodar, Ahmad Babaeian, "Knowledge Extraction and Inference Based on Visual Understanding of Cooking Contents" (2021). *USF Tampa Graduate Theses and Dissertations*.
<https://digitalcommons.usf.edu/etd/9278>

This Dissertation is brought to you for free and open access by the USF Graduate Theses and Dissertations at Digital Commons @ University of South Florida. It has been accepted for inclusion in USF Tampa Graduate Theses and Dissertations by an authorized administrator of Digital Commons @ University of South Florida. For more information, please contact digitalcommons@usf.edu.

Knowledge Extraction and Inference Based on Visual Understanding of Cooking Contents

by

Ahmad Babaeian Jelodar

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy in Computer Science and Engineering
Department of Computer Science and Engineering
College of Engineering
University of South Florida

Major Professor: Yu Sun, Ph.D.
Shaun Canavan, Ph.D.
Heather Culbertson, Ph.D.
John Licato, Ph.D.
Kyle Reed, Ph.D.

Date of Approval:
June 30, 2021

Keywords: Knowledge Representation, Video Understanding, Deep Learning, Ingredient
Recognition, Calorie Estimation, Dish Classification

Copyright © 2021, Ahmad Babaeian Jelodar

Dedication

To my dear parents that have guided me through the years and to my dear wife for being there
for me.

Acknowledgments

"Life is the warmth of the intertwined hearts,

All gates are locked if no friends inside."

—Fereydoon Moshiri

I must first acknowledge and thank my family, my mother, my father, and my brother Mostafa, and my sister Marzieh, for their constant support and guidance. I must thank my dear wife for her unconditional support and being there for me without expectation.

I must thank many friends at USF. To my lab mates at the Robot Perception & Action Lab, I must thank Yongqiang (a.k.a. Garfield), John, Troi, David, Mohammed, Tianze, Salekin, Juan, Hailey, and Maxat. Thank you for all the social conversations that we held in the lab. Next, I must thank fellow students that I've met along the way: Anwesh, and Pablo for their companionship through the years.

Next, to the faculty and staff here at USF: I sincerely thank Dr. Yu Sun, my advisor, for guiding me through my journey here. You have taught me to consider logical reasoning and thinking no matter the obstacles and biases. I thank my committee members who have all played a role in the development of our research work: (in alphabetical order) Dr. Shaun Canavan, Dr. Heather Culbertson, Dr. John Licato, and Kyle Reed. I am also grateful to the staff members in our department office: Laura Owczarek, Jessica Pruitt, and Mayra Morfin and former members of the CSE department Gabriela Franco, and Lashanda Lightbourne (a.k.a. Shanie). I appreciate you all and continue to rely on you all to help us stay on track.

At this moment, I want to thank everyone who I could not mention by name. By no means are your contributions invalid to my success nor do I disregard your support, and I take with me all of the experiences we've shared along the way. Thank you.

Table of Contents

List of Tables	iv
List of Figures	v
Abstract	vii
Chapter 1: Introduction	1
1.1 Related Works	3
1.1.1 Knowledge Representation	3
1.1.2 Image Classification and Object Detection	5
1.1.2.1 Image Classification	5
1.1.2.2 Multi-label Image Classification	6
1.1.2.3 Object Detection	6
1.1.3 Video Understanding	9
1.1.3.1 Raw Video Understanding	9
1.1.3.2 Knowledge Representation for Video Understanding	10
1.1.4 Understanding Cooking States	10
1.1.5 Cooking Applications	11
1.1.5.1 Dish Classification	11
1.1.5.2 Ingredient and Recipe Recognition	12
1.1.5.3 Portion and Calorie Estimation	13
1.2 Contribution of Dissertation	14
1.3 Structure of Dissertation	15
Chapter 2: Video Understanding	17
2.1 Functional Object-Oriented Network	18
2.1.1 FOON Basics	18
2.1.2 Functional Unit	19
2.1.3 FOON Construction	20
2.1.4 FOON Sources and Statistics	20
2.1.5 FOON vs. Other Knowledge Representations	21
2.2 Video Understanding Pipeline	22
2.2.1 Functional Object Recognition	23
2.2.2 Functional Motion Recognition	24
2.2.3 Functional Unit Recognition	25
2.2.4 Task Graph Inference	25
2.3 Functional Object Recognition	25
2.3.1 Recognizing Objects-in-action	25

2.4	Functional Unit Recognition	27
2.4.1	Functional Unit Confidence	27
2.4.2	Probing	29
2.5	FOON for Video Understanding	30
2.5.1	Object Overlap Metric	31
2.5.2	Functional Unit Recognition Analysis	32
2.5.2.1	Functional Unit Recognition Using FOON	32
2.5.2.2	Functional Unit Recognition with Motion and FOON	33
2.5.2.3	Analysis	35
2.5.3	Video Understanding	35
2.5.4	Task Inference (Recipe Classification)	37
2.5.5	Discussion	39
2.6	Conclusion and Future Work	40
Chapter 3:	Understanding Cooking States	41
3.1	The State Identification Challenge	42
3.1.1	The Challenge	42
3.1.2	The States Identification Dataset and Statistics	44
3.1.3	The Dataset Details	45
3.2	Baseline State Analysis	46
3.2.1	Experimental Baseline Analysis	47
3.2.1.1	State Identification	48
3.2.1.2	ImageNet Test	49
3.2.1.3	State Analysis	50
3.2.2	Imagenet Analysis	51
3.3	Joint Object and State Recognition	52
3.3.1	Stage 1: Double Loss Convolutional Network	52
3.3.2	Stage 2: Language Knowledge Based Features	53
3.3.2.1	Feature Extraction	54
3.3.3	Stage 3: Neural Network Predictions	55
3.3.4	Stage 4: Model Refinement	55
3.3.5	Results and Analysis	56
3.4	Generating Multiple States and Ingredients	57
3.4.1	State and Token Embeddings	58
3.4.1.1	States Exploration	58
3.4.1.2	Embeddings	60
3.4.2	Ingredients Given States	61
3.4.3	States Generation Analysis, Experiments and Results	63
3.4.3.1	Dataset	63
3.4.3.2	Implementation Details	64
3.4.3.3	Experiments on State Embeddings and Prediction	64
3.4.3.4	Experiments on Ingredient Prediction Given States	65
3.5	Conclusion and Future Work	67

Chapter 4:	Meal Image Understanding and Analysis	70
4.1	Transformer Decoders	71
4.2	Ingredient Generation	73
4.2.1	Dish Classification	74
4.2.2	Ingredient Generation	76
4.2.2.1	Formulation	77
4.3	Ingredient Portion and Calorie Estimation	77
4.3.1	Individual Ingredient Analysis	78
4.3.2	Simultaneous Ingredient Analysis	79
4.3.2.1	Inputs and Intermediate Outputs	80
4.3.2.2	Encoder Structure	80
4.3.2.3	The Final Network	81
4.3.2.4	Losses	83
4.4	Dataset Preparation for Analysis	83
4.4.1	Unit Identification and Sampling	85
4.5	Experiments and Analysis	86
4.5.1	Implementation Details	86
4.5.2	Results	86
4.5.2.1	Ingredient Generation	88
4.5.2.2	Portions and Units Estimation (Meal Kits)	89
4.5.2.3	Calorie Estimation	90
4.6	Discussion and Future Work	92
Chapter 5:	Concluding Remarks	94
5.1	Future Work	95
References		97
Appendix A:	The Ingredient, Portions, and Calorie Estimation Web Interface	107
A.1	Browsing and Selecting an Image	107
A.2	Dish Type Selection	108
A.3	Main Ingredient Generation	108
A.4	Optional Ingredient Generation	109
A.5	Seasonings Generation	109
A.6	Per Ingredient Portion and Calorie Estimation	110
Appendix B:	Copyright Permissions	111

List of Tables

Table 2.1	Results of probing objects	30
Table 2.2	Top 10 accuracy for functional unit recognition	34
Table 2.3	Recipe classification results.	39
Table 3.1	Baseline classification accuracy on the state dataset	48
Table 3.2	Baseline state classification accuracy per ingredient fine-tuning	49
Table 3.3	Baseline classification accuracy of the Imagenet subset	50
Table 3.4	Simultaneous states and object classification accuracy	57
Table 3.5	Ingredients and their states extracted from the Recipe1M dataset.	59
Table 3.6	States prediction results	65
Table 3.7	Ingredients generation results given states.	67
Table 4.1	Intersection over union (IoU) for ingredient generation	89
Table 4.2	Main ingredient prediction results	89
Table 4.3	MAE portion estimation for different units	90
Table 4.4	Calorie estimation results	92

List of Figures

Figure 1.1	Illustration of a simple image classification model.	5
Figure 1.2	Illustration of a simple multi-label image classification model.	7
Figure 1.3	Illustration of an object detection model.	8
Figure 1.4	Illustration of dish classification and ingredients generation models.	12
Figure 1.5	Illustration of models for recipe retrieval and portion and calorie estimation.	13
Figure 2.1	Illustration of a functional unit with input and output object nodes.	19
Figure 2.2	Illustration of universal FOON.	21
Figure 2.3	The pipeline for automatic functional unit identification.	23
Figure 2.4	Illustration of active objects identification.	26
Figure 2.5	Illustration of the functional unit confidence estimation.	28
Figure 2.6	Illustration of how overlap is measured.	32
Figure 2.7	Precision and recall for functional unit recognition.	33
Figure 2.8	Example of functional unit recognition analysis.	33
Figure 2.9	Snapshots of qualitative analysis of functional units.	36
Figure 2.10	Results of overlap metrics for video understanding.	37
Figure 2.11	Calculated F-scores for video understanding.	38
Figure 2.12	Illustration of identified functional units for noodle recipe.	38
Figure 3.1	An illustration of the state taxonomy.	44
Figure 3.2	An illustration of the dataset class statistics.	45
Figure 3.3	Example images of eleven classes of the dataset.	46
Figure 3.4	Proposed baseline states network.	47
Figure 3.5	Confusion matrices of the state classification results.	51

Figure 3.6	Samples of mis-predicted images on the states baseline model.	52
Figure 3.7	Samples of multi-state or ambiguous images in Imagenet.	53
Figure 3.8	Pipeline for simultaneous state and object recognition.	54
Figure 3.9	Illustration of the pipeline refinement module.	56
Figure 3.10	Objects and states CNN probabilities vs Concept-Net probabilities.	58
Figure 3.11	Image to states and ingredients generation example.	59
Figure 3.12	Recipe1M states statistics.	60
Figure 3.13	Two-step model for ingredient prediction.	62
Figure 3.14	Recall, precision and F1-score of states.	66
Figure 3.15	Samples of states and ingredient prediction.	68
Figure 4.1	An illustration of meal understanding.	71
Figure 4.2	The two stage pipeline for calorie estimation.	72
Figure 4.3	Original and modified versions of the transformer architecture.	73
Figure 4.4	Details of the transformer model.	74
Figure 4.5	Ingredients generation models.	75
Figure 4.6	Illustration of individual ingredient and simultaneous ingredient labeling.	76
Figure 4.7	Individual ingredient analysis model.	78
Figure 4.8	The calorie estimation model.	82
Figure 4.9	Dish types in Recipe1M.	85
Figure 4.10	Ranges of values for most frequent units in the Recipe1M dataset.	87
Figure 4.11	All measurements originally available in the Recipe1M dataset text.	88
Figure 4.12	Examples of results of the end-to-end meal image understanding pipeline.	91

Abstract

In this dissertation, we discuss our work on analyzing cooking content for the ultimate goal of automatic robotic manipulation. For a robot to perform a cooking task, it will need to both have an understanding of the scene and utilize prior knowledge. We will explore two main sub-problems: *knowledge extraction and inference*, and *visual understanding of the scene* in this dissertation. Visual understanding of a scene, requires algorithms that can visually infer information from a single image or video. Many algorithms in the area of image classification, object detection, or activity recognition can be used in this area. Although great advances has been achieved by the emergence of deep learning, state-of-the art algorithms in this area have limitations. To attempt to overcome this lack of performance, we propose to use structured knowledge representations combined with state of the art deep learning techniques for visual understanding of cooking videos. Besides objects, and motions, we recognize that states of objects are also very important in interpreting the scene and therefore extensively explore the problem of states in visual cooking content. We introduce the state identification challenge in cooking applications and collect a dataset for research in the area of ingredient state analysis. We further look into the problem of simultaneous knowledge extraction from a single image and extracting information about ingredients, their states, the inter-connection between different objects in the scene and the motion-object interconnections. This problem requires an algorithm that can model the correlation of various concepts in a single image simultaneously. Using deep algorithms that can take as input multiple inputs and generate multiple outputs are fit for this problem. Therefore we propose to incorporate auto-regressive self-attention based mechanisms to extract knowledge from a single image. We show that the knowledge acquired from a single image can be used for calorie estimation. We suggest that total knowledge extraction from a single image can be used in future work for task graph inference.

Chapter 1: Introduction

¹ Robotic manipulation and its applications has significantly increased in the past years. It has been applied or researched in various applications such as medical, agricultural, industrial, sports, collaborative based robots, and domestic applications. One of the applications that has recently gained attention in robotics is the cooking domain for both industrial and domestic settings. In this dissertation, our aim is to analyze problems associated with a cooking robot and what it requires to understand the environment and manipulate.

For a robot to act automatically in an environment, it has to initially look at the environment using sensors (e.g. visual or depth cameras), interpret what it has seen using advanced algorithms (e.g. state of the art image and video understanding algorithms), and manipulate accordingly. In this dissertation we focus on understanding what a robot observes and what steps we require to extract structured knowledge from the scene and also from any given cooking content to use for robotic training and automatic scene understanding. The raw data from robotic sensors is usually in the format of image streams (i.e. videos) hence we need to use algorithms that can parse images and videos to perform interpretations of the scene. Because our focus is on an automatic cooking robot we focus our attention towards understanding scenes and videos with a cooking setting.

To analyze a scene, a robot first has to understand what *objects* (i.e. ingredients) it identifies in the video. For example, if the robot identifies the ingredient *tomato* in the video, it would know that the video is demonstrating a cooking activity that requires tomato. Therefore, one of the problems we look into in this dissertation is object (i.e. ingredient) recognition. Multi-label image classification and object detection algorithms can be helpful in reaching to a solution for this step of the problem. Not only the robot needs to identify the ingredients, it also needs to determine the *state* of the ingredient. Recognizing states helps to both extract knowledge from videos and

¹This chapter was partially published in [1, 2, 3]. Permission is included in Appendix B.

also to understand the scene. For example if the robot finds a *whole tomato* but requires a *sliced tomato* to make *omelette*, it would need to take action and perform *slicing* to prepare the tomatoes for its usage. But, if the robot already identifies the tomato as sliced it would not require any further steps to prepare the tomato. In this dissertation, we extensively look into the problem of visual state identification for objects in a cooking setting. Besides ingredients and their states, *portions* of ingredients (e.g. 1 cup tomato) are also crucial in making a recipe. You can not make an *omelette* without knowing how many *eggs* or how much *spinach* you will be using. Portions can be determined using visual estimations and identifying correlations between ingredient features and visual features.

To extract knowledge from video we also need to understand each event (i.e. motion) happening in the video. Many motions (e.g. slicing) are directly related with state change in an object (e.g. tomato) and determining either would help identify the other. But some motions (e.g. pouring) would not necessarily reflect any physical state change in any ingredient and therefore may need a separate encoder to identify. In this dissertation we also create a model for motion recognition in a video and show that it is important for cooking event recognition.

Ingredients and their states in a meal are not separate entities. Conversely, they are correlated entities although some ingredients might not be visually recognized. For example, assuming an image of *baked cake*, if we know that the meal uses *flour* in its recipe, with high probability we can say it uses *sugar* although it is not visually recognizable. Therefore, to better model each of the ingredients, states and portions we need to take into account their mutual relationships and build associations between them either through knowledge representation or model training. We propose to learn the relationship between objects, actions, and activities and represent those relationships in a graph. We use the graph as structured prior information for video understanding when possible. For example, a video that demonstrates a chef who is cooking an omelet comprises multiple consecutive actions, and each action, such as mixing eggs in a bowl, employs multiple objects such as a bowl, a whisk, and eggs. To identify the actions, the structural information between the objects (bowl, whisk, eggs) and motions (mixing) is useful. For instance, if we understand that *eggs* can be mixed using a *whisk*, we can associate the object *whisk* with the objects *egg* and *bowl*.

The structural information between consecutive actions can also be applied to interpret an activity in a video. For example, cracking eggs into a bowl occurs before mixing the eggs in the bowl. Consequently, this knowledge can help with predicting that the current ongoing action is mixing, knowing that the previous action was cracking eggs. Embedding these informative structures into a prior graphical structure and using the embedding for inference at test time can improve video understanding.

Our aim in this dissertation is to extract the required knowledge from visual content (i.e. cooking content) such as ingredients, states, portions and the motions associated with the activity demonstrated in the video. To do that we incorporate both known knowledge representations and also execute machine learning based algorithms to create models that can encode knowledge about cooking videos and manipulation scenes. We use the extracted knowledge to create structured and sequential data in the form of a task graph for the robot to be able to use it.

1.1 Related Works

We now discuss research works that have developed algorithms for understanding visual content both in image and video in a broad range and in cooking applications. We first will take a look at some of the work that use knowledge representation mainly for visual content analysis and further review the works on computer vision and image processing especially in cooking applications.

1.1.1 Knowledge Representation

The field of knowledge representations is an important domain in the robotics field. However, there is a lack of a formal definition on what is categorized as a knowledge representation. The concept of knowledge representation was first introduced in the field of Artificial Intelligence (AI) and is "concerned with how knowledge can be represented symbolically and manipulated in an automated way by reasoning programs" [4]. This definition emphasises representation of knowledge through logical expressions and lacks focus for inference from knowledge. In [5] they define an extension to knowledge representation for robotic applications as a "means of

representing knowledge about a robot's actions and environment, as well as relating the semantics of these concepts to its own internal components, for problem solving through reasoning and inference". Therefore a knowledge representation contains information about how objects and motions related to the robotics application are associated with each other and a robot can use that to perform tasks and manipulations.

Knowledge representations have been successfully applied to robotics and machine learning [6] and in natural language processing for Wordnet [7], Verbnet [8], and Framenet [9]. WordNet is a freely available large lexical database of English Nouns, verbs, adjectives and adverbs and are grouped into sets of cognitive synonyms (synsets), each expressing a distinct concept. Synsets are associated with each other through semantic and lexical edges. Wordnet has been used in many computational linguistic applications and it is helpful to associate parts of speech together [10]. VerbNet is a hierarchical network of English verbs that maintains syntactic and semantic edges between them [8].

In [11], Carlson et al. propose a knowledge-based architecture to learn a language from web text. Some works introduce and use a knowledge base for answering queries [12], visual queries [13], and cuisine- and ingredient-oriented queries using deep features [14]. Knowledge-based methods have been also used in visual applications such as the ontological hierarchical knowledge base for image content retrieval and video event detection [15], scene understanding [16], description logics for scene interpretation [17], visual structured knowledge base for scene recognition and object detection [18], and a combination of various knowledge based representations using machine learning and statistical approaches [19]. In [20], the problem of object affordance reasoning is modeled using a knowledge base representation. In [21] a visual knowledge-based representation and dataset are introduced for modeling relationships in images. In [22], a knowledge representation-based method for food recognition from an image was proposed, which is close to our application. The lack of a structured knowledge representation for joint object and motion representation motivated applying the functional object-oriented network for video understanding in cooking videos.

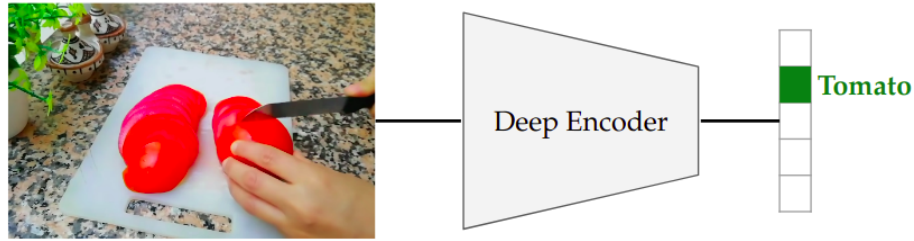


Figure 1.1. Illustration of a simple image classification model which takes as input an image and returns as output a vector of confidences (probabilities) per class.

1.1.2 Image Classification and Object Detection

Images are valuable sources of information. In recent years many deep learning algorithms have been proposed to extract useful knowledge from images and perform tasks such as image classification and object detection for various tasks (e.g. human detection, car detection). In this sub-section we review some of the work proposed in this area that can be used as base algorithms for knowledge extraction from cooking content.

1.1.2.1 Image Classification

In image classification applications, the model takes as input as single image and produces as output a single class label [23, 24, 25] as shown in Figure 1.1. The assumption in these types of models is that there is one dominant object (i.e. ingredient) in the image. Most common image classification models use deep learning approaches. Deep convolutional networks have dominated the image classification field from 2012 after the Alexnet paper [26] was presented. Deeper models such as VGG [23] were proposed thereafter were they can capture deeper and richer features from the image and provide good performance of image based classification tasks. Other models such as Inception [24], and Resnet [25] have also proposed optimizations to deep convolutional networks to accommodate for different filter sizes in a layer and better gradient propagation in deeper models. We in our work use variants of Resnet for training, fine-tuning or extracting features from images whenever needed.

1.1.2.2 Multi-label Image Classification

In image multi-label classification applications, the assumption is that the model takes as input as single image and produces as output multiple labels [27] as shown in Figure 1.2. In other words, multi-label classification models are models that can jointly model various concepts in a single image simultaneously.

Some models propose to combine CNN with an RNN for multi-label classification [27]. The CNN derives semantic features from an image and the RNN models associations between the image and the labels and the labels themselves. The labels in this type of a model have an ordered relationship. In [28] a model has been proposed to maximize subset accuracy using recurrent neural networks. In [29] the authors propose a Canonical Correlated AutoEncoder (C2AE), for multi-label classification by performing joint feature and label embedding and introducing a label-correlation sensitive loss function to recover the predicted label outputs.

Image captioning models also fall into the category of multi-label classification. Image captioning models first create a vocabulary of words of size N and assign ids between 0 to $N - 1$ to each of the words [30]. The word ids are converted to one-hot embeddings and used as output for model prediction [31]. In image captioning models the labels are generated one step at a time which is different from multi-label classification models which are created jointly. Also, image captioning models require an ordered dependency between the generated outputs which not present in multi-label classification outputs. Both of these types of models produce multiple labels for a single image. Image captioning models can usually utilize auto-regressive models. Many of the models use recurrent neural networks (e.g. LSTM) [32, 33], and some of them use transformer based and attention based mechanisms as proposed in [34]. In [35], they propose an ingredient prediction model based on the transformer model which outperforms the state-of-the-art multi-label classification model with target distribution loss in [36].

1.1.2.3 Object Detection

In object detection applications, the assumption is that the model takes as input as single image and produces as output multiple bounding boxes with their class labels [37, 38, 39] as

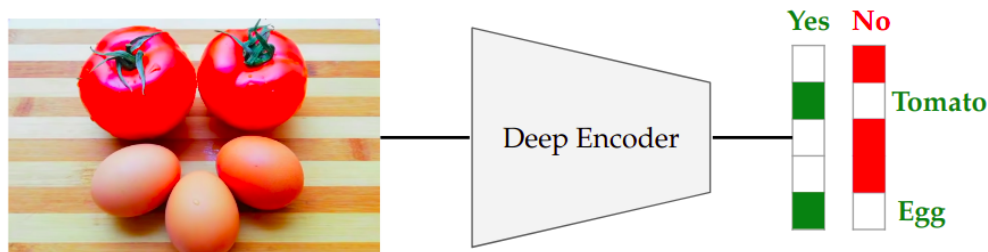


Figure 1.2. Illustration of a simple multi-label image image classification model which takes as input an image and returns as output a binary classification confidence per class name.

shown in Figure 1.3. State of the art object detection methods have also followed the path of image classification algorithms and use mainly deep convolutional networks as base for feature extraction and simultaneous object localization and labeling.

Faster R-CNN first takes an input image to ConvNet and returns feature maps of the image. From there it applies a Region Proposal Network (RPN) on the feature maps to extract object proposal for the given image. All the proposals are further resized to the same size and passed to a fully connected convolution layer. Bounding boxes are classified to object labels. Therefore results provided for each image are a list of bounding boxes and their labels.

Single Shot MultiBox Detector (SSD), as its name suggests, runs a convolution network for an input image and computes the feature map for that image [37]. Then it runs a small 3x3 sized convolution network on the feature map to predict the bounding boxes of its relative category. SSD also uses anchor boxes at a variety of aspect ratios comparable to Faster R-CNN and learns the off-set to a certain extent than learning the box. In order to hold the scale, SSD predicts bounding boxes after multiple convolutional layers. Since every convolutional layer functions at a diverse scale, it is able to detect objects of a mixture of scales [37].

We present YOLO, a new approach to object detection. Prior work on object detection repurposes classifiers to perform detection. Instead, we frame object detection as a regression problem to spatially separated bounding boxes and associated class probabilities. A single neural network predicts bounding boxes and class probabilities directly from full images in one evaluation. Since the whole detection pipeline is a single network, it can be optimized end-to-end directly on detection performance. Our unified architecture is extremely fast. Our base YOLO model processes

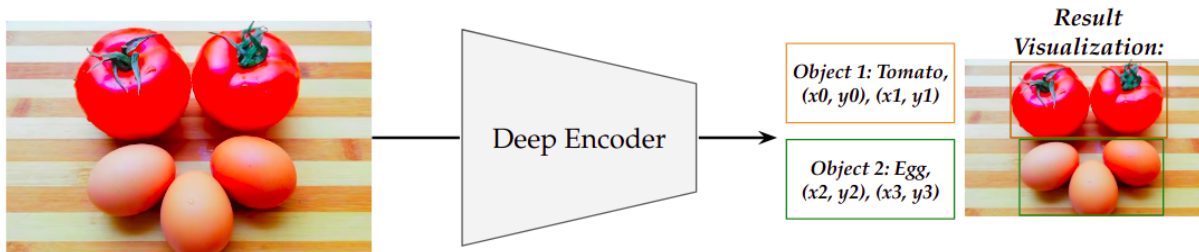


Figure 1.3. Illustration of an object detection model which takes as input an image and returns bounding boxes and their class names.

images in real-time at 45 frames per second. A smaller version of the network, Fast YOLO, processes an astounding 155 frames per second while still achieving double the mAP of other real-time detectors. Compared to state-of-the-art detection systems, YOLO makes more localization errors but is less likely to predict false positives on background. Finally, YOLO learns very general representations of objects. It outperforms other detection methods, including DPM and R-CNN, when generalizing from natural images to other domains like artwork

In YOLO [40], object detection is formulated as a regression problem through unified detection as a one stage object detection network. In unified detection, the image is first split into a grid and each box in the grid is associated with multiple potential bounding boxes and confidences. YOLO works as a single neural network model and performs on full images in one evaluation and can be trained end-to-end. The unified detection mechanism makes YOLO very fast but the version one model does not have a good performance in comparison to other object detection methods.

Retina-net is a one-stage object detection network that at the time it was proposed performed better than all two-stage networks (e.g. Faster R-CNN) and could match around the same speed of one-stage networks [38]. In Retinanet, the authors proposed a novel loss named Focal loss that addresses the issue of class imbalance in the object detection problem. Focal loss focuses attention on harder samples and prevents the easy samples from overwhelming the optimization procedure. Although YOLO, SSD and Retinanet are faster algorithms, but Faster RCNN performs better than those algorithms with more advanced CNN bases.

1.1.3 Video Understanding

1.1.3.1 *Raw Video Understanding*

There is a broad area of work in video understanding. Some works deploy costly setups such as physical sensors or additional modalities (e.g., text) [41, 42, 43], and some research performs analysis on spatio-temporal features of a sequence in a holistic manner to label actions [44, 45, 46, 47] or uses spatio-temporal features of a person (e.g., models of joints or pose) to classify actions [48, 49]. These methods are incapable of handling variations in view, zoom, and occlusion easily. Simultaneous video segmentation and understanding [50], [51], [52] is also a very common research area. These methods usually do not consider objects or variations in pose. Some approaches extract and analyze a selection of frames for video event summarization [53], and fast anomaly concentration and detection [54]. Jain et al. propose a method that embeds structure into a deep model [55] to incorporate knowledge with deep models for activity recognition. Other deep approaches proposed for activity recognition are [56, 57]. The motivation to incorporate FOON for video understanding is based on the group of research that use objects and their affordances and states in a video for action recognition [58, 59, 60, 61, 62, 63].

Currently, there are various multi-view applications, especially in surveillance systems. Information from multiple cameras can enhance event summarization or task understanding. Several researchers have proposed methods for handling multi-camera scenarios. Event summarization in multi-view videos using a deep learning approach [57], detection and summarization of an event in multi-view surveillance videos by applying boosting [64], and a machine learning ensemble method [65] are instances of research in the area of multi-view video understanding. This aspect of video understanding has not been addressed in this work; however, the proposed framework can be deployed in multi-view systems. A discussion on the multi-view aspects of our video understanding pipeline is included in Section 2.5.5.

1.1.3.2 *Knowledge Representation for Video Understanding*

Various approaches have been proposed to use knowledge representation for video understanding, such as semantic-visual knowledge bases like FrameNet and Imagenet for modeling rich event-centric concepts and their relationships for video event detection [66], a knowledge- and probabilistic-driven framework for activity recognition [67], and semantic representations for event detection [68, 69]. Souza et al. deploy objects, actions and their bonds into graphs and use simulated annealing for event inference using temporal connections [70], [71]. Ren et al. [72] previously proposed a Bayesian framework that uses object motions and their relationships to improve object recognition reliability. This model enables robots to learn the interactive functionalities of objects from human demonstrations [73] [74].

Object information and analysis is an essential aspect for activity recognition. The method in [75] deploys spatial and functional constraints on the relationships between objects and motions to semantically interpret videos. Modeling the mutual context of human pose and objects using a random field model [76], modeling relationships between object parts and people in the scene using contextual scene descriptors and Bayesian learning [77], and encoding objects for action classification and localization are examples of work on video understanding using object information. These works all assume that a person is performing the act in the video, and, therefore, the human pose would be essential for their approaches. We follow the path of incorporating objects and extend it to the goal of action recognition and activity inference by deploying our previously proposed knowledge representation network [78]. Our work is different from the noted object-based activity recognition methods, in that our videos do not contain a person and its pose. We use only the human hand and its location, if available in the scene, as features to interpret the video.

1.1.4 Understanding Cooking States

To the best of our knowledge, no specific work has been done in the area of image state identification. In this section, we discuss work in the area of image classification, image captioning, and understanding that are relevant to or motivated us for this research. Currently, image classification has shifted towards convolutional neural networks. Krizhevsky et al, introduced the first

evolutionary deep model for image classification [26]. Thereafter, other deep models such as VGG [23], Googlenet [79], and Resnet [25] were introduced gradually as deeper and more advanced networks for image classification. Improvements with the combination of these networks have also has been introduced in [80]. These works all focus on image object classification and do not consider states of objects in an image.

In [81], the authors show the importance of using object parts in recognizing an action from an image, thereby modeling human actions with a parts and attributes base. This work is an obvious proof of how object parts and states can help recognize an object or understand an image. Work such as [30] and [31] provide captions for images or videos. In [30] Yao et al. use attributes and their interactions with deep networks to provide captions. Other work such as [33], and [82] perform multi-label classification on a single image using RNN- and CNN-based deep architectures. Although these papers provide various labels for an image, they do not consider states of objects as another label for the image. These papers have one thing in common – they analyze an image to understand it. The state identification problem, also motivated by this aspect, contributes to the understanding of images.

Some work has been conducted in the area of cooking images and videos. Food recognition systems for dietary analysis [83, 84], fruit recognition [85], and ingredient recognition for recipe retrieval [86] are instances of work, which focus on recognizing or detecting the ingredients in an image. Some papers, such as [87, 88], perform food recognition on video to understand the whole video and associate it with a recipe or action. Other papers, such as [89, 90], focus on activity recognition from cooking videos. These works contribute to understanding cooking images and videos, but none explicitly focus on states. To our knowledge, we are the first to address this problem in cooking images or videos.

1.1.5 Cooking Applications

1.1.5.1 *Dish Classification*

Dish (or food) classification can be considered as an application of image classification. Some work such as [91, 92] provide experimental studies on small scale datasets to recognize food

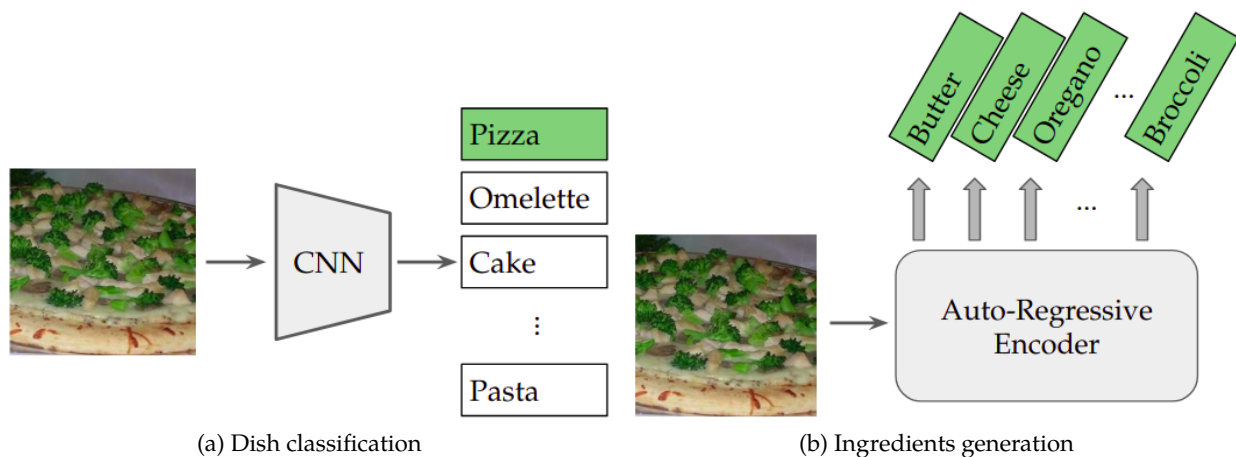


Figure 1.4. Illustration of dish classification (returns one label per image) and ingredients generation models (returns multiple labels per image).

(or dish) types from a single given image. Many applications of dish classification incorporate non-visual context such as geo-location to increase dish classification accuracy [93, 94]. In [95], to address the dynamic and changing nature of food and dish classification, Horiguchi et al. proposed a personalization based model for dish classification. The commonality between the proposed work in this area is a small scale dataset and a deep learning model to address that. We in this paper create our own set of dish types and use a state of the art deep learning network to perform dish classification.

1.1.5.2 Ingredient and Recipe Recognition

Research in the area of ingredient recognition can be classified into two main categories: retrieval based, prediction. In retrieval based applications, a list of ingredients or the whole recipe is retrieved based on creating an embedding and retrieving the appropriate image match from the dataset [96, 97, 98, 99] (as shown in Figure 1.5.a).

This body of work requires the predicted combination of ingredients to be a fixed set as seen in one of the datasets. To handle this issue, ingredient prediction approaches inspired by multi-class modeling [27, 28, 29], recurrent image captioning [30, 31, 33], and auto-regressive list prediction methods emerged [100, 35]. Ingredient state recognition is also another field of study that has been under-studied. Introducing new ingredients states datasets [101, 102], or addressing the states

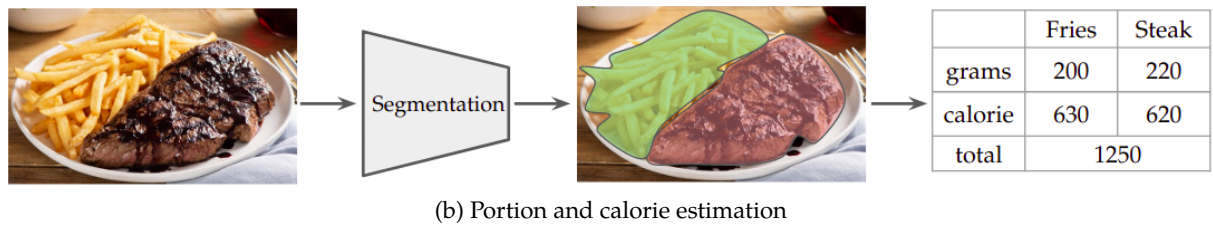
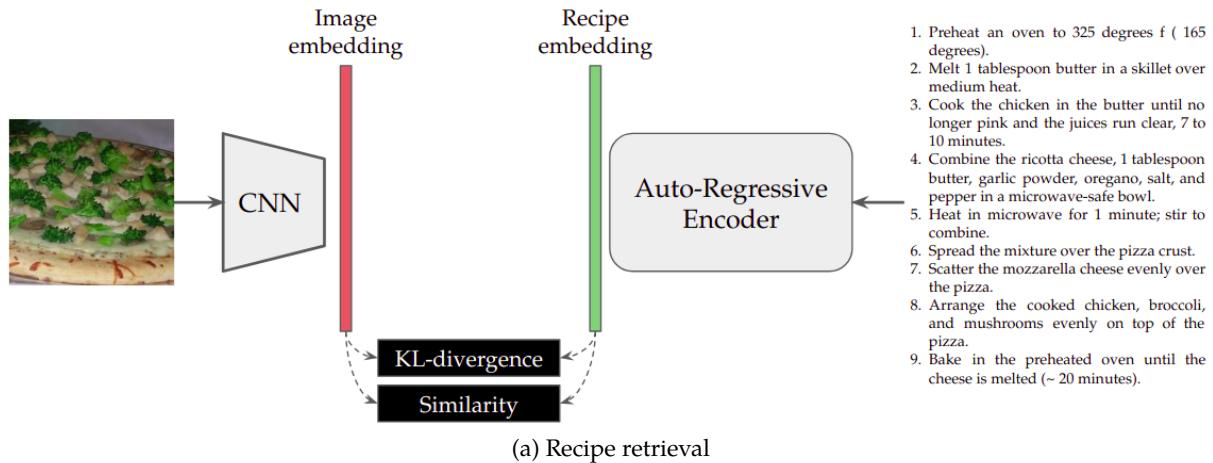


Figure 1.5. Instances of models for applications in the cooking domain - models for recipe retrieval and portion and calorie estimation.

problem as image classification or multi-class labeling (i.e. ingredient-state tuple) problem [101, 1] are instances of research in this area.

One recent work that we use as our baseline for ingredient prediction is the inverse cooking research [35]. In this work, both ingredients and recipe are generated in auto-regressive manner using the transformer model [35].

1.1.5.3 Portion and Calorie Estimation

Research on ingredient portions has mainly been conducted in the context of calorie estimation. In most of the work [103, 104, 105, 106] portions of ingredients (e.g. apple) are identified after image segmentation [103] (as shown in Figure 1.5.b) and size computation [107] using various approaches (e.g. geometry, 3d modeling) for calorie estimation of very simple cooking images in small scale datasets [105]. Also, approaching portions from a visual recognition and segmentation view may not be feasible in meals where the ingredients is not visually discerned (e.g. chicken in

soup). Therefore we approach the portions problem in a self-attention query based manner using a large scale dataset (i.e. Recipe1M).

Calorie estimation from image has gained attention in food and image processing research. Some methods propose multi-stage pipelines to predict food categories/ingredients, identify portions/sizes and estimate the calorie intake of the food [103, 104, 105, 106]. Some of these methods take two input images to define depth and segmentation of food in the image [103, 105]. These algorithms use model based or deep learning based methods for the recognition stage and standard nutritional fact tables for calorie estimation [106]. Some literature directly provide estimates of calorie from a food image [108, 109] by predicting the food category itself and directly mapping it to a calorie intake for that meal with [109] or without a reference object. The drawback to these methods is that they do not take into account the variety of ingredients different versions of a meal can have. Some work propose a CNN-based direct image method that take into account multiple food in one image but still do not consider the containing ingredients of meals for calorie estimation [110, 111, 112]. Also, in [113] authors propose a food-estimation Bayesian framework for food-balance estimation which considers a limited number of food categories with a limited number of classes each with limited discrete values.

Most of the work done in the area of calorie estimation assumes that images are of food with clear segmented boundaries [103, 105, 114] and do not consider addressing more complex food such as mixed or cooked meals where the containing ingredients are not clear. Another issue with these work is that the dataset used is very small and low diversity and the images are captured in a well-controlled setting. On the other hand, there are a few literature that exploit ingredients for image-based calorie estimation [115]. In [115] a deep learning based method is proposed for simultaneous learning of calories, categories, ingredients and cooking directions. Datasets such as Japanese calorie-annotated food photo dataset and the American calorie-annotated food photo dataset [115] are datasets with calorie annotations.

1.2 Contribution of Dissertation

This research work contributes the following to the robotics community:

1. We propose a model to encode object-object affordances, and motion-object associations for improving video understanding in cooking videos.
2. We use the structural information of knowledge representations for task inference using object-object and object-motion associations.
3. We defined the state identification problem in cooking for fine-grained activity understanding and created a taxonomy of cooking states. Using the taxonomy we proposed a dataset of cooking states for robot manipulation.
4. We propose a model that provides improved ingredient recognition by integrating ingredient states into deep convolutional networks. We also propose a model for joint object and state recognition using deep convolutional networks and knowledge representations.
5. We propose the simultaneous portions estimation model for recognizing ingredients, portions and calories simultaneously for visually and non-visually apparent ingredients using deep query based self-attention mechanisms.
6. We propose a deep self-attentive model for calorie estimation in large scale.
7. We propose a multi-level framework for ingredient generation.

1.3 Structure of Dissertation

This dissertation is structured as follows:

- In Chapter 2, we introduce a four stage pipeline for video understanding from cooking content using the FOON knowledge representation [78, 2].
- In Chapter 3, we analyze states and their changes in cooking images and introduce the state identification challenge. Alongside the challenge, we introduce a dataset of 10,000 images with their ingredient and state labels. We also propose models for single state recognition, joint state and object recognition, and set based state recognition using deep models on our own dataset and large scale datasets such as Recipe1M [99].

- In Chapter 4, we delve into knowledge extraction given a single image. We focus on extracting information such as list of ingredients, their states, portions, and per ingredient calorie based on a single image shot. We suggest that this type of information extraction can be used for task graph generation and side applications such as total calorie estimation.
- In Chapter 5, we end our discussion with concluding remarks on the state of research in cooking content analysis for the purpose of robotic manipulation and what future work holds for this path of research.

Chapter 2: Video Understanding

² The initial challenge to automatic robotic manipulation, is understanding the scene. To understand the scene, robots need to visually analyze the scene and interpret entities for further decision making. Video understanding is the main general solution to this challenge. Video understanding is a challenging topic that requires completion of several difficult steps successfully, where each step is a challenging and active research topic by itself. It would usually require the video to be automatically split into atomic actions, the activities and objects in the atomic video clip to be successfully recognized, and a meaningful understanding inferred based on the activities and objects. For each step, extensive learning would be carried out for object recognition, activity recognition, and video splitting, but these are usually done individually.

We propose to learn the relationship between objects, actions, and activities and represent those relationships in a graph. We use the graph as structured prior information for video understanding when possible. For example, a video that demonstrates a chef who is cooking an omelet comprises multiple consecutive actions, and each action, such as mixing eggs in a bowl, employs multiple objects such as a bowl, a whisk, and eggs. To identify the actions, the structural information between the objects (bowl, whisk, eggs) and motions (mixing) can be useful. For instance, if we understand that eggs can be mixed using a whisk, we can associate the object whisk with the objects egg and bowl.

The structural information between consecutive actions can be applied to interpret an activity in a video. For example, cracking eggs into a bowl occurs before mixing the eggs in the bowl. Consequently, this knowledge can help with predicting that the current ongoing action is mixing, knowing that the previous action was cracking eggs. Embedding these informative structures into

²This chapter was partially published in [3]. Permission is included in Appendix B.

a prior graphical structure and using the embedding for inference at test time can improve video understanding.

We use the coordination encoded in the object nodes (e.g., bowl or eggs) and motion nodes (e.g., stirring) of the knowledge-based graph presented in [78, 2] to recognize actions (such as stirring eggs) in videos. The knowledge-based network used for task inference, called the *functional object-oriented network* or FOON [78], encodes knowledge about the flow of actions coming one after another. Using this network, we present a powerful object-oriented inference algorithm for action and activity recognition.

We propose a pipeline that deploys object localities and their motion features to identify active objects within an action. We train a deep model for holistic motion recognition, which helps with cases in which the object (e.g., salt in a chef’s hand) is not easily detectable. The identified objects and motion are fed to the inference stage with FOON to provide a list of candidate functional units that can be associated with the current ongoing action (e.g., cracking egg in a bowl). The consecutive predicted functional units are evaluated to understand the activity performed in the video (e.g., making an omelet).

2.1 Functional Object-Oriented Network

Functional Object-Oriented Network (FOON) is a knowledge representation for encoding knowledge about manipulation tasks and, in extension, object affordances. A FOON can also be used by a robot for solving manipulation problems given a target goal. Currently, FOON focuses on learning activities in the cooking and kitchen domain, but it can also be extended to other domains and environments.

2.1.1 FOON Basics

A FOON is a directed acyclic graph that contains two types of nodes (object and motion), making it a bipartite network [116]. Figure 2.1 depicts a sample functional unit, the basic building block of a FOON.

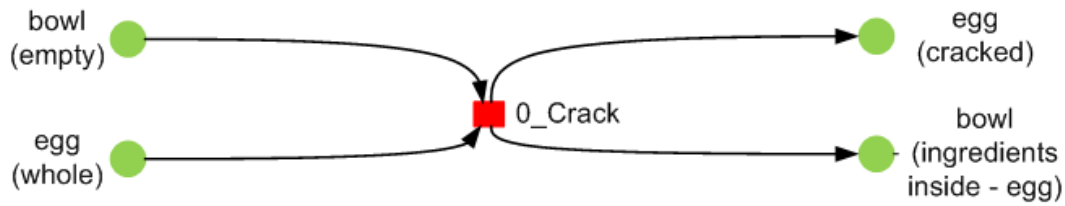


Figure 2.1. Illustration of a functional unit with input and output object nodes, their states, and a motion node.

Object nodes are defined as items that are being manipulated or acted upon by a demonstrator, and motion nodes describe the action being applied on objects such as cutting or mixing. An object node (N_O) is identified by its *object type*, an *object state*, and a *motion identifier*, which denotes whether the object is in motion during activity. Objects can also serve as *containers* of other objects, and each node can be described by a list of ingredients. Motion nodes are identified only by their *motion type*. Within this graph, as in regular bipartite networks, edges connect a pair of nodes; specifically, an edge in FOON connects an object-motion pair. The edge direction indicates the order in which an object may change in its state through a motion action similar to Petri Nets [117], which require transitions to activate or fire place nodes.

2.1.2 Functional Unit

A FOON consists of subcomponents or learning units called functional units. Each functional unit describes a single, atomic action as seen in an activity (an activity or subgraph can be considered as a series of actions). For instance, in the activity of cooking scrambled eggs, one functional unit may describe the action of cracking an egg, and another may describe the action of mixing the eggs in a bowl. A functional unit describes the transition of objects' states before and after a manipulation motion occurs; this is described by input object nodes (objects before manipulation) and output object nodes (objects after manipulation). In this paper, our focus is generating these functional units directly from instructional videos for learning future instances of how tasks are executed. A collection of subgraphs (or activities) that are merged together to combine knowledge and remove duplicate units is called a *universal* FOON. Each functional unit has three components: input object nodes, output object nodes, and a motion node that describes the action that possibly

causes a change in the input objects' states, possibly causing a state change, because an action may not always incur a change of state. Each functional unit is also described by the time frames at which they are observed in an activity.

2.1.3 FOON Construction

The graph shown in Figure 2.2 consists of nodes from 65 videos that were annotated in the form of subgraphs, which consist of functional units that reflect each individual step in a cooking procedure. Edges would be drawn between an object node and motion node pair, where the object nodes are those seen in an action within the cooking activity and the motion node describes the action occurring. As we created these subgraphs and parsed them, we compiled a list of objects and motions to create labels for the different node instances seen and to enforce consistency in labels (subgraphs were created by multiple volunteers). When adding new information (subgraphs) from other datasets, we only need to annotate them to conform to the format of our graphs and parse them to get the labels correct. The merging procedure will add these newly parsed functional units to the network to ensure that there are no duplicates. This merging procedure is detailed more in our previous work in [78]. This is where this proposed work fits in; the task of automatically generating subgraphs from videos (especially those from other datasets) is difficult to do, and manual annotation can be time-intensive.

2.1.4 FOON Sources and Statistics

A FOON ideally is learned directly from human demonstrations, whether by video or from observation, and it is automatically generated from such demonstrations, although in the earlier phases of constructing FOON we opted to manually label YouTube videos as subgraphs. In the future, we will try to extend FOON using the method discussed in this paper. After recording all functional units for a video, we parsed the subgraph to ensure that all object and motion labels were consistent with all other subgraphs.

Each subgraph was then merged into a single network, referred to as a *universal FOON*. The merging procedure is as follows: using a list of all functional units in G_{FOON} , compare each

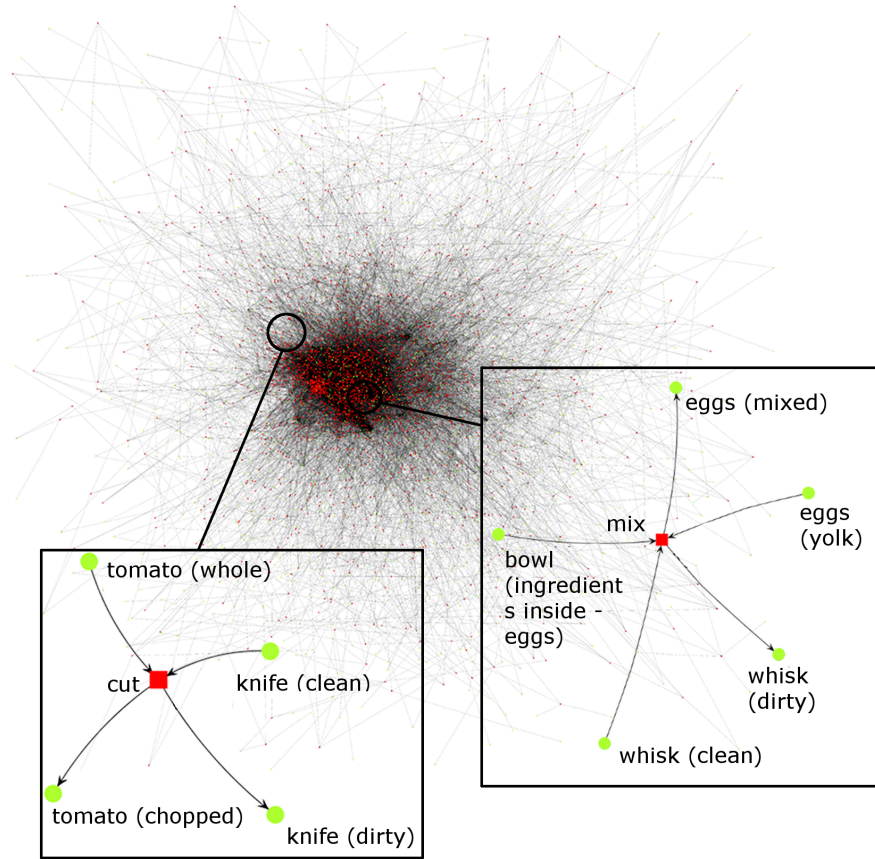


Figure 2.2. Illustration of universal FOON with 4955 nodes (object and motion nodes). FOON comprises many functional units, such as those highlighted in image.

functional unit in all subgraphs to this list and append those units of a subgraph which are not present in G_{FOON} .

In total, the network contains 1853 object nodes, 3102 motion nodes, and 15656 edges. Figure 2.2 illustrates the network described by these statistics.

2.1.5 FOON vs. Other Knowledge Representations

FOON is not the first knowledge representation to address video understanding. In this subsection, we discuss the main differences between FOON and previous work. Previous works in knowledge representation do not consider the joint representation of both objects and motions. Our work is inspired by the theory of affordance originally proposed in [118]. Many follow-up

studies show that there is a link between manipulations and objects. Our objective is to create a graphical representation of manipulations where objects and motions describe affordance. In terms of graphical representations, previous works capture knowledge using probabilistic graphical methods or semantic graphs/trees. However, they do not create a knowledge base of activity from demonstrations that could then be used for performing (possible) new manipulations. In addition, for affordance studies, they would instead try to model the relationship between objects and simple actions to predict the effect or impact it has on them. A more general form of representation akin to FOON is Petri Nets, where place nodes are like object nodes and transition nodes are like motion nodes. Certain input places are needed to “fire” or execute a transition node, much like input object nodes must be available to execute a given manipulation motion.

2.2 Video Understanding Pipeline

We propose a four-stage pipeline for video understanding. The pipeline identifies the objects and motions in a video sequence (associated with an action) and uses them together with the knowledge representation to assign a functional unit label to the event in action. An *action* refers to a single, atomic event, and a *sequence* of actions represents an entire activity. Consecutive identified actions will be analyzed as a whole to understand the activity (recipe) being executed in the video. The steps to the pipeline are as follows: 1) *functional object recognition*, 2) *functional motion recognition*, 3) *functional unit recognition*, and 4) *task graph inference*.

In the first stage of the pipeline, the *functional object recognition* stage, all objects are identified and scores are assigned to objects based on their usefulness in the scene. In the second stage, *functional motion recognition*, each action (a split of the video) is classified into its corresponding motion class. Using the results from the first two stages and their FOON accordances, each action is analyzed and associated with a functional unit in the *functional unit recognition* stage. Each recognized action (in the video) from a single video is assigned a functional unit and looked up in the FOON graph for them to eventually be classified as a whole activity (recipe). This last stage is referred to as *task graph inference*. An illustration of the video understanding pipeline is depicted in Figure 2.3.

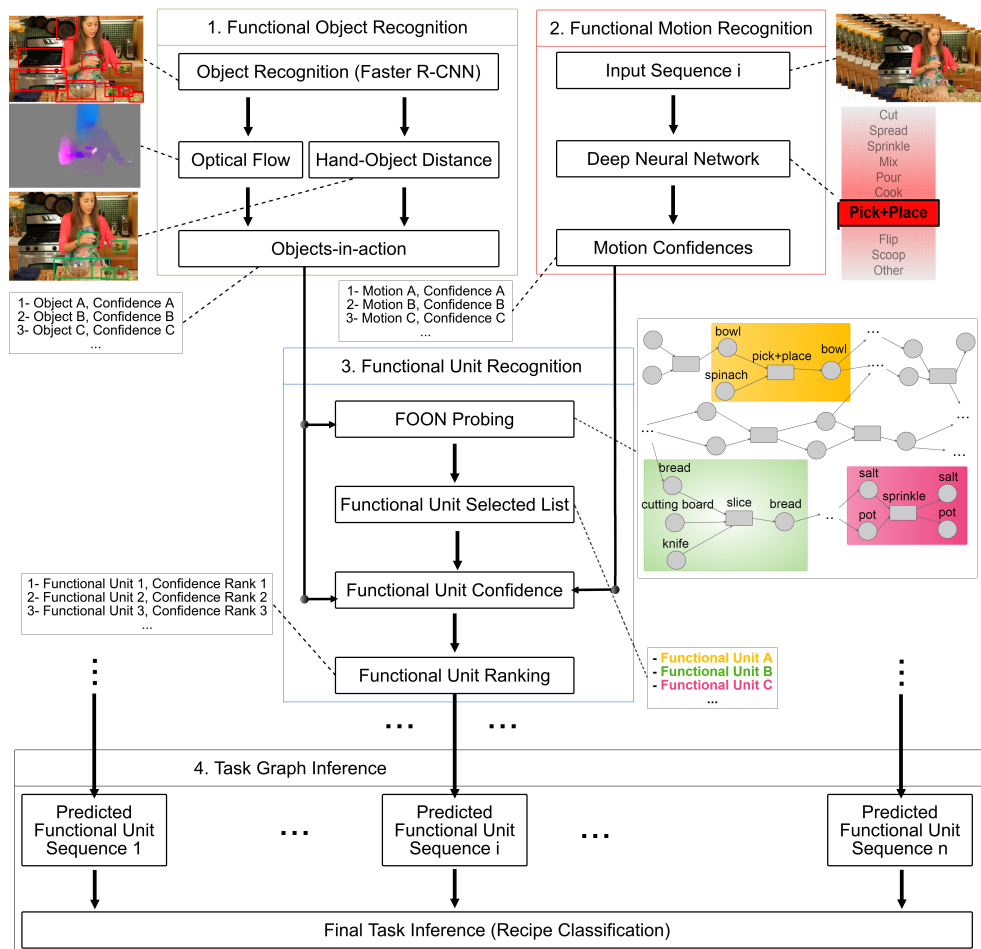


Figure 2.3. The four stage pipeline for automatic video understanding including 1) functional object recognition, 2) functional motion recognition, 3) functional unit recognition, and 4) task graph inference.

2.2.1 Functional Object Recognition

We apply the well-known Faster R-CNN algorithm for localizing and labeling objects in the scene [39]. Faster R-CNN is a two-part convolutional network that detects object proposals and performs object classification simultaneously. The output of the Faster R-CNN network is a set of bounding boxes and their corresponding object class labels. We further identify the used objects in the video sequence, which we call objects-in-action, using three metrics—the closeness of the human hand to the object, the magnitude of optical flow, and the frequency in which the

objects have been observed in the video. We explain the functional object recognition stage more thoroughly in Section 2.3.1.

2.2.2 Functional Motion Recognition

In some cases, FOON is not able to correctly identify the action in video using only object features. For example, knowing that the objects *bowl* and *egg* are objects-in-action could lead to multiple FOON inferences, because various functional units contain the object nodes *bowl* and *egg* but have different motion nodes (e.g., pouring or cracking). In another example, when sprinkling salt with the hand, it is difficult to visually discern that the object *salt* is being used, but the hand motion will suggest the action of sprinkling.

To address these issues, we fine-tune the deep (CNN+LSTM) network by Donahue et al. [45] with 10 classes in the last layer. This network comprises a CNN portion and an LSTM portion. The frames of a sequence are, one by one, given as input to the CNN and the output of the CNN is given as input to the LSTM layer. The outputs of the LSTM layer are averaged to provide a final prediction for the class of the motion in action. The architecture of the CNN network contains five convolutional layers and two fully-connected layers. The initial five convolutional layers and a single fully-connected layer on top is fed to one layer of a recurrent LSTM layer. The output of the LSTM layer is followed by the classification layer. We modified the last layer so that the number of neurons in the last layer of the network contains ten neurons to reflect the 10 motion types we have selected for training. We train the CNN architecture and the CNN + LSTM architectures separately. We use the trained weights from [45] and perform training only for the last layer of classification. We report only the better results from the CNN+LSTM architecture.

Each motion class in this deep architecture is associated with a set of motion nodes in FOON. The network assigns confidence scores to each of the motion classes. A confidence score reflects the probability of a class being assigned as a label to the action happening in the video. For more details on the approach, readers are referred to the algorithm described in [45]. The output from this deep network is used to calculate confidences for each candidate functional unit in the functional unit recognition stage.

2.2.3 Functional Unit Recognition

We determine the meaning of a video by associating actions with functional units. Objects-in-action are looked up in the universal FOON to identify candidate functional units. Candidate functional units are evaluated based on a confidence score computed in this stage and thoroughly discussed later in Section 2.4.1. This consolidated confidence score incorporates both object confidences produced from the functional object recognition stage and motion confidences resulting from the functional motion recognition stage. The confidence score estimates how related each candidate functional unit is to the ongoing action in the present sequence. The list of candidate functional units is further sorted based on their confidences. Functional units with the highest confidences are associated with the current action.

2.2.4 Task Graph Inference

To identify the activity (sequence of actions) in a video, the identified actions throughout the video are used together with FOON look-up to predict the most likely activity label for that video.

2.3 Functional Object Recognition

We recognize and localize all objects in a video sequence (associated with an action) using the well-known Faster R-CNN algorithm [39]. We then quantify the involvement of each object in the current action by extracting optical flow features and calculating hand-object distances in each frame of the video sequence. A list of the most used objects is created and named objects-in-action.

2.3.1 Recognizing Objects-in-action

In this stage of the pipeline, we use the bounding box associated with each object for our computations. After localizing objects, the less frequent objects in the video are excluded. The center point of the bounding boxes resulting from the Faster R-CNN algorithm are used to calculate the *object's average distance from the hand*. The distances are further normalized using a Gaussian distribution. The *optical flow of objects* within the video sequence are computed. The proposed

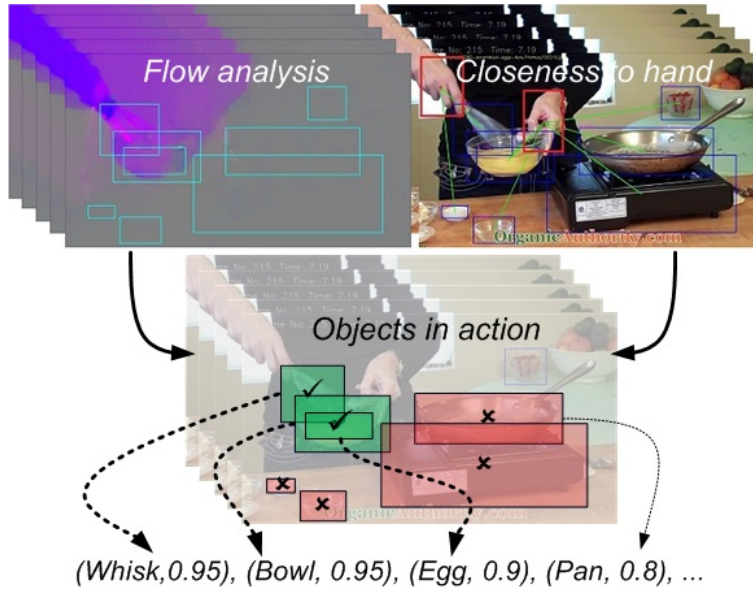


Figure 2.4. Example showing procedure of identifying objects-in-action. Items such as *egg* and *whisk* would be possible candidates for participating in an egg-whisking motion.

method in [119] is used to estimate the optical flow between two frames. The estimated optical flow and the objects' positions are incorporated to estimate the flow of each object. Objects with higher magnitude of optical flow are assigned a higher confidence value—A higher value conveys a higher chance that the object is moving and, hence, a higher probability that the object is being used in the video sequence. Equation 2.1 shows how these metrics are integrated to estimate a confidence for each object.

$$conf_{object} = \alpha \cdot c_{flow} + \beta \cdot c_{dist} + \gamma \cdot c_{freq} \quad (2.1)$$

where c_{flow} , c_{dist} , and c_{freq} are the optical flow confidence, distance to hand confidence, and frequency confidence of each object, respectively; $conf_{Object}$ is the final calculated confidence of the object. Coefficients α , β and γ are tuned manually and represent how much each factor contributes to the final confidence of each object. Figure 2.4 depicts the procedure of identifying objects-in-action for a simple action of whisking eggs, using the three metrics mentioned in Equation 2.1. In the example of Figure 2.4, we observe an egg-whisking motion occurring in which objects *egg*,

whisk, and *bowl* are at the top of the list of objects-in-action and objects *pan* and *stove* have lower confidences.

2.4 Functional Unit Recognition

Each action in the video is associated with the closest functional unit from FOON. To associate the correct functional unit with an action, unrelated functional units are filtered out. Filtering is performed using *functional unit confidence* estimation and *probing*, as discussed in this section.

2.4.1 Functional Unit Confidence

The pipeline recommends a list of in-use objects from the current action, named objects-in-action (Section 2.3.1). Objects from the list are looked up in FOON, and functional units containing them are identified. The identified functional units are suggested as candidate functional units that can be associated with the current action in the video. Every functional unit contains several object nodes that may or may not be included in the list of objects-in-action. The overlap between the object nodes (of a functional unit) and the objects-in-action are named as the used set, and the remainder of the object nodes is tagged as the unused set. These two sets of objects are used to determine whether we should support or penalize a candidate functional unit. Equation 2.2 shows how the confidence of a candidate functional unit is estimated.

$$conf_{FOON} = \frac{\sum_{n=1}^{N_{used}} conf_n}{N_{used}} - penalty + \kappa.bonus \quad (2.2)$$

In this equation, $conf_{FOON}$ is the estimated confidence, N_{used} , is the number of object nodes in the used set of a candidate functional unit, and $conf_n$ is the confidence of each of those objects (Section 2.3.1). The *bonus* term is estimated based on the pixel-wise overlap of all objects used in a candidate functional unit. This term represents the extent of interaction between the objects. The *penalty* term calculated by Equation 2.3, represents the penalty applied to the estimated confidence.

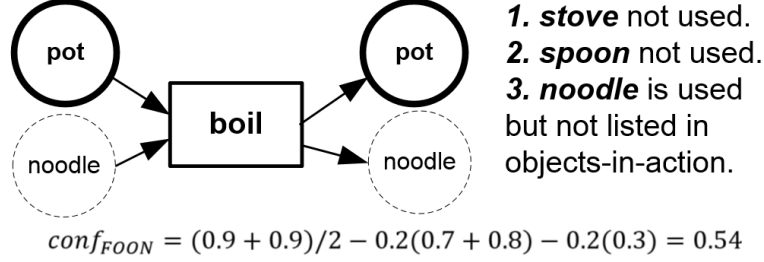


Figure 2.5. Illustration of functional unit confidence estimation. In this example, identified objects-in-action are *pot*, *spoon*, and *stove*, with confidences 0.9, 0.8, and 0.7 respectively ($\lambda=\eta=0.2$).

$$penalty = \sum_{m=1}^{N_{notused}} \lambda \cdot conf_m + \sum_{k=1}^{N_{extra}} \eta \cdot conf_k \quad (2.3)$$

The confidence of the objects listed in the list of objects-in-action but not used in the candidate functional unit, $conf_m$, together with the confidence of the objects not listed as objects-in-action but used in the candidate functional unit, $conf_k$, contribute to the penalty. In this equation, $N_{notused}$ is the number of unused objects, and N_{extra} is the number of objects not listed but used in the candidate functional unit. In Equation 2.2, the constant κ tunes the effect of bonus and penalty. The constant λ in Equation 2.3 tunes the effect of unused objects on the penalty term, and the constant η tunes the effect of objects used but not listed. Figure 2.4.1 illustrates the procedure of confidence estimation for a candidate functional unit. The algorithm for confidence calculation is shown in Algorithm.

The confidence calculated in Equation 2.2 focuses solely on object interaction and their functional affordances. We believe motion can introduce additional information for confidence calculation. To include functional motion for estimating the confidence, we incorporate the outputs from the trained deep architecture (CNN+LSTM) for motion classification. We fuse the output of the CNN+LSTM network with the confidence estimated solely based on object interaction, $conf_{FOON}$. The output of the CNN+LSTM network for motion recognition has 10 confidence scores representing the probability of each of the motion classes happening. We rank the motion classes based on their resulted confidence scores. Finally, the confidences of functional units in Equation 2.2 are combined with the results from the CNN+LSTM network to extract a final confidence for the functional units as shown in Equation 2.4.

Algorithm 1: Confidence Calculation

```
1: list =  $\emptyset$  // holds objects and their confidences
2: for object  $\in$  sequence do
3:    $c_{dist} = \text{abs}(\text{object} - \text{hand})$ 
4:    $c_{freq} = \text{frequency}(\text{object})$ 
5:    $c_{flow} = \text{opticalFlow}(\text{object})$ 
6:    $\text{conf}_{object} = \alpha \cdot c_{flow} + \beta \cdot c_{dist} + \gamma \cdot c_{freq}$ 
7:   list.append((object,  $\text{conf}_{object}$ ))
8: end for

9: list.sort()
10: topK = list.selectTopK() // objects in action
11: // Find all candidate functional units associated with the top K objects
12: candidates = FOONLookUp(topK)

13: for c  $\in$  candidates do
14:   nodes = c.getObjects()
15:   overlap = objectOverlap(nodes.objects, topK.objects)
16:    $N_{used} = \text{size}(\text{overlap})$ 
17:   bonus = pixelOverlap(nodes.objects)
18:   unused = (topK - overlap) + (nodes - overlap)
19:   penalty = average(unused.confidences)

20:    $\text{conf}_{FOON}(c) = \frac{\sum_{n=1}^{N_{used}} \text{conf}_{object}(n)}{N_{used}} - \text{penalty} + \kappa \cdot \text{bonus}$ 
21: end for
```

$$\text{conf}_{motion} = \text{conf}_{FOON} + \alpha \cdot \text{conf}_{LSTM} \quad (2.4)$$

In Equation 2.4, conf_{FOON} is the confidence calculated in Equation 2.2, and conf_{LSTM} is the confidence calculated based on results from the CNN+LSTM network. Coefficient α balances the effect of each of those parameters.

2.4.2 Probing

Each object is individually looked up in FOON, and all functional units containing that object are identified. A list of candidate functional units containing the object is acquired. The list contains candidate functional units that may associate with the current action. We exclude the objects with lower confidences, conf_{object} , from the list, to reduce the number of potential objects-

Table 2.1. Results of probing objects in FOON based on the example in Section 2.4.2. The objects-in-action are shown in bold.

	Input Nodes	Motion	Output Nodes	Overlap
1	mixer, bowl	mix	mixer, bowl	0.5
2	fork, egg , cup	stir	fork, egg , cup	0.67
:	:	:	:	:
674	bowl , pan, pasta	pour	pan	0.25

in-action and, as a consequence, the number of probed objects and candidate functional units. To illustrate, assume that the filtered list of objects seen in the sequence or probed objects are *egg*, *bowl*, and *fork* and the ground truth functional unit associated with the sequence that includes the motion node *mix* with the objects *bowl*, *egg*, and *fork* as input nodes and *egg*, and *fork* as output nodes. Individually probing functional units in FOON using the list of objects produces a list of candidate functional units that contain those objects. Table 2.1 shows some of the 674 candidate functional units that contain the objects-in-action for this specific example. Any other functional unit that is not identified does not contain the objects.

Each probed functional unit from FOON contains object nodes that may or may not have been observed in the current video sequence (associated with an action). The last column of Table 2.1 depicts the overlap between the objects included in a probed functional unit with the identified objects in the video sequence. The probed functional units with an overlap value less than a specific threshold are excluded. Confidence values for the remaining functional units are computed, and those with the highest confidence values are selected. The selected functional units are the most likely to be associated with the ongoing action.

2.5 FOON for Video Understanding

For analyzing the power of the Functional Object Oriented Network for video understanding we perform some experiments and analyze the results. In our experiments, we used the annotated videos used for the creation of the universal FOON in [78] and the videos from the MPII Cooking Activities Dataset, summing to a total of 338 videos [120]³.

³The videos and graphs of FOON are available at: <http://www.foonets.com/>

At the time of writing this dissertation, the universal FOON consisted of data from 338 instructional videos and a total of 3102 functional units. This includes a subset of instructional videos from YouTube and videos from the MPII Cooking Activities Dataset [120]. For the current experiments, we also manually labeled some of the video sequences in FOON with object bounding boxes and their categories.

We used 11 of the 338 cooking videos as our test dataset, which included an overall amount of 55 functional units. We performed our tests in 11 iterations in a leave-one-out manner: in each iteration, one video was entirely left out, and the remainder of the videos were used to create a FOON. The lack of training data for object recognition, lack of labeled ground truth data for the videos, and a shortage of instances of each kind of functional unit in the dataset forced us to only use 11 videos for the experiments.

For testing the pipeline, we conducted three different experiments based on both manually- and automatically-labeled objects. 1) comparing functional unit recognition using only FOON look-up with functional unit recognition using the fusion of FOON and motion recognition, 2) video understanding for functional unit recognition with and without FOON, and 3) task inference or recipe classification.

2.5.1 Object Overlap Metric

The overlap between a candidate functional unit and its corresponding ground truth functional unit was used to evaluate the results. This overlap metric was calculated for each action in the video separately. The metric used was fairly simple: if the motion node of the candidate functional unit was equivalent to the motion node of the ground truth functional unit, the overlap between their object nodes was counted. Consequently, precision and recall were computed using the object overlap. *Precision* was measured as overlap divided by the number of object nodes in the candidate functional unit, and *recall* was measured as overlap divided by the number of object nodes in the ground truth functional unit. If the motion nodes were different, precision and recall were assumed to be 0. Figure 2.6 illustrates how precision and recall were calculated.

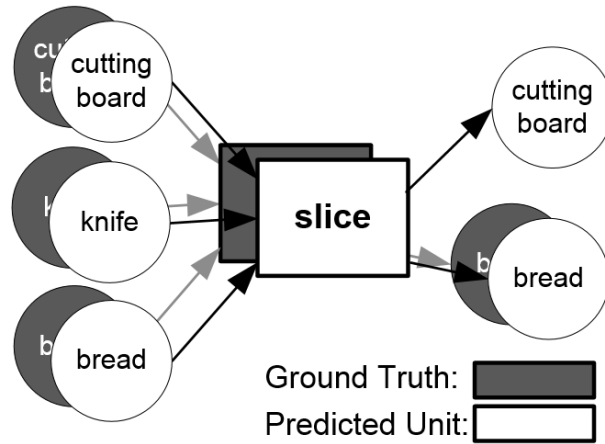


Figure 2.6. Illustration of how overlap is measured. In this example, overlap is equal to 4 nodes, precision is 100 percent and recall is 80 percent. If the ground truth motion node was anything but *slice*, precision and recall would be 0 percent.

2.5.2 Functional Unit Recognition Analysis

We used the time stamp labels in the universal FOON to split the videos in the dataset into its comprising actions. For example, a video demonstrating a cook making scrambled eggs was split into several atomic actions such as cracking eggs, pouring eggs into a bowl, and mixing eggs with a whisk.

2.5.2.1 Functional Unit Recognition Using FOON

Each action sequence in a video was fed into the algorithm that identified the functional unit best fit for that action based on the metrics discussed in Section 2.4. In each iteration, we used a single video for evaluation and the other 337 videos to create an iteration-specific FOON. Functional units corresponding to an action sequence in a video were identified by processing the iteration-specific FOON. After identifying functional units, precision and recall were computed as defined in Section 2.5.1 for all candidate functional units for the top 10 results, as shown in Figure 2.7.

In Figure 2.7, the horizontal axis represents the number of best functional units analyzed for precision and recall calculation. The solid curves show precision, and the dashed curves show recall calculated on 55 functional units for both manually- and automatically-labeled objects.

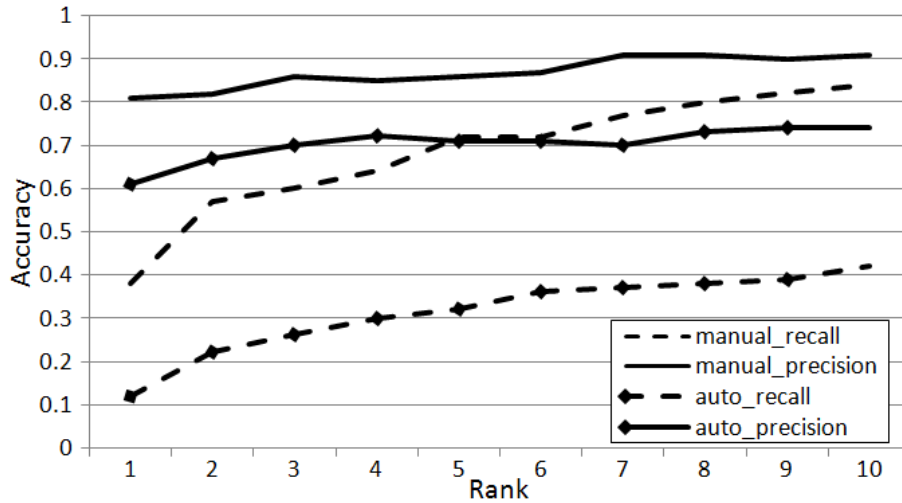


Figure 2.7. Precision and recall, as observed in manual and automatic object recognition for top 10.

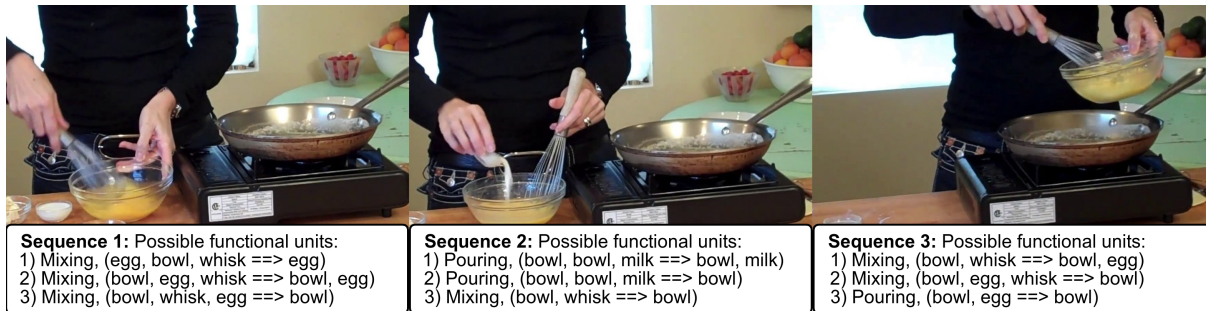


Figure 2.8. Example of functional unit recognition using labeled and manually-split sequences for scrambled egg recipe.

Figure 2.7 shows that the algorithm can potentially improve with additional procedures. We can also see that precision in Figure 2.7 is always higher than 80 percent, showing that our algorithm sometimes missed the objects in the video; however, when it assumed an object was being used in a functional unit, it usually identified the functional unit correctly. In Figure 2.8 snapshots of three sequences of a cooking video are depicted with their predicted functional units. In this example, the correct functional unit is always included in the top three identified functional units.

2.5.2.2 Functional Unit Recognition with Motion and FOON

We fused motion recognition with FOON look-up to improve the recognition procedure. We created motion classes by selecting the nine most frequent motion types from the FOON motion

Table 2.2. Top 1 to 10 accuracy of prediction for functional unit recognition using FOON and Motion Recognition.

	Using FOON	Using FOON + Motion Recognition
Top 1	56%	64%
Top 3	75%	84%
Top 5	80%	89%
Top 10	89%	98%

nodes (e.g., *pour*, *pick+place*, and *cook*) [78]. To accommodate the other types of motions not included in the nine most frequent motions, we designed a class labeled as the other class. We extracted optical flow features from each sequence in the video, applied the CNN+LSTM network on RGB and optical flow sequences of each event, and performed an averaging of the outputs from the two networks. The architecture returned 10 values representing confidences for the 10 classes. The motion confidence values were used in the computation of confidences for the candidate functional units. Table 2.2 shows the top 1, top 3, top 5, and top 10 accuracy of prediction for functional unit recognition using both FOON and motion recognition.

The accuracy of prediction for an action was computed by comparing the identified functional units with the ground truth functional units. If the motion node of the identified functional unit was equivalent to the motion node of the ground truth and the overlap of object nodes was higher than 80 percent, we determined the prediction to be correct. We counted the number of correct predictions over all functional units in the test set and calculated the accuracy. In some cases, the motion node of the ground truth varied in text with the motion node of the identified functional unit, while having the same interpretation (e.g., *whip* vs. *stir* or *slice* vs. *cut*). These cases of motion nodes were considered equivalent.

As shown in Table 2.2, the accuracy of functional unit recognition when motion recognition was combined with FOON look-up was higher than functional unit recognition without motion recognition. This shows adding automatic motion recognition to the pipeline improves the motion node recognition and leads to better identification of functional units. The deep network guesses the motion node in only 47 percent of the cases. The complexities of the videos, such as background variations, different camera views, and moving cameras, prevented it from producing the desired

accuracy. In experiments, we set α in Equation 2.4 to less than 0.2, so the results from the neural network would not adversely influence the final results.

2.5.2.3 *Analysis*

To see the effect of each part of the pipeline on the results, we looked deeper into each part. The automatic motion recognition by itself achieves 67 percent accuracy, whereas the functional unit recognition without motion recognition achieves 61 percent accuracy (top 2). There are two differences in these two evaluations that make them incomparable. First, for automatic motion recognition, the number of classes of motion was generalized and reduced to 10 classes, whereas for functional unit recognition, there were more than 50 types of motion nodes. Second, functional unit recognition identifies the action with a focus on both the objects and the motion occurring, whereas the aim of motion recognition is to recognize the motion class in an action. Although not comparable, motion recognition is a good feature to combine with FOON for optimal functional unit recognition.

We calculated the overlap between objects-in-action and the identified functional units as 84 percent. This shows that although the majority of objects were identified correctly, the accuracy of functional unit recognition was lower than expected due to mistakes in identifying the motion nodes.

In another experiment, we applied the pipeline combined with motion recognition for automatically recognized objects and report its top 10 results in Figure 2.7. Although object recognition is an important stage of the pipeline that can be improved, we do not address it further, as that is not our specific goal in this dissertation. Snapshots of various sequences with their ground truth representation and identified functional units are depicted in Figure 2.9.

2.5.3 Video Understanding

The pipeline was evaluated based on the extent it understands a video using the overlap metric. Precision and recall were calculated for both object and motion nodes for all actions of each video

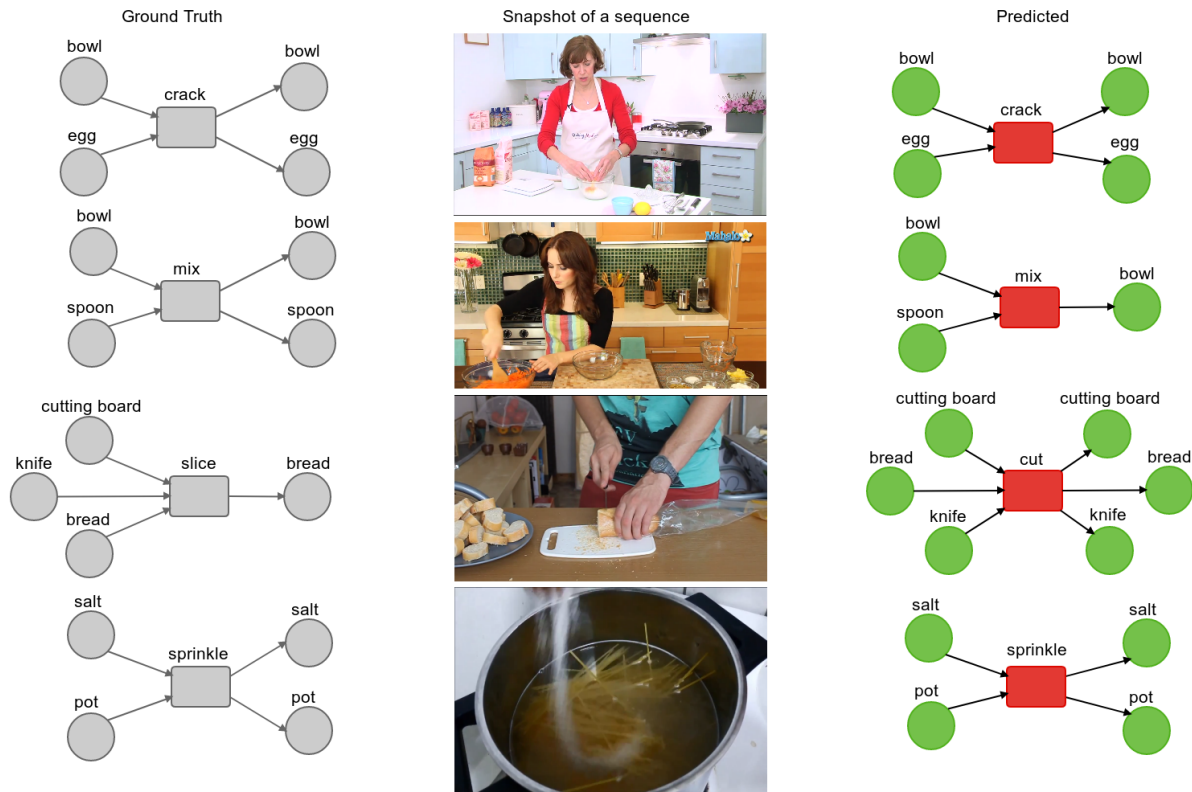


Figure 2.9. Snapshots of events, their ground truth functional unit representation and predicted functional unit.

individually, and the average precision and recall was calculated for all videos over the top 10 results. Figure 2.10 shows the calculated results.

The results show that the pipeline is capable of perceiving an understanding of the video, especially when the top 5 results are used. The lower values for recall may be due to errors made in identifying objects-in-action. We calculated the F-Score metric using recall and precision, as discussed in [121].

The video understanding F-Score was calculated for the pipeline in two instances: 1) when FOON was used, and 2) when FOON was not used; the results are depicted in Figure 2.11. When using FOON, we calculated the F-Score by using the overlap metric for ground truth and identified functional units. When not using FOON, we calculated the overlap metric between the highest-ranked objects and the objects in the ground truth functional unit, and we calculated the overlap between the highest-ranked motion classes with the motion nodes in the ground truth. The sum

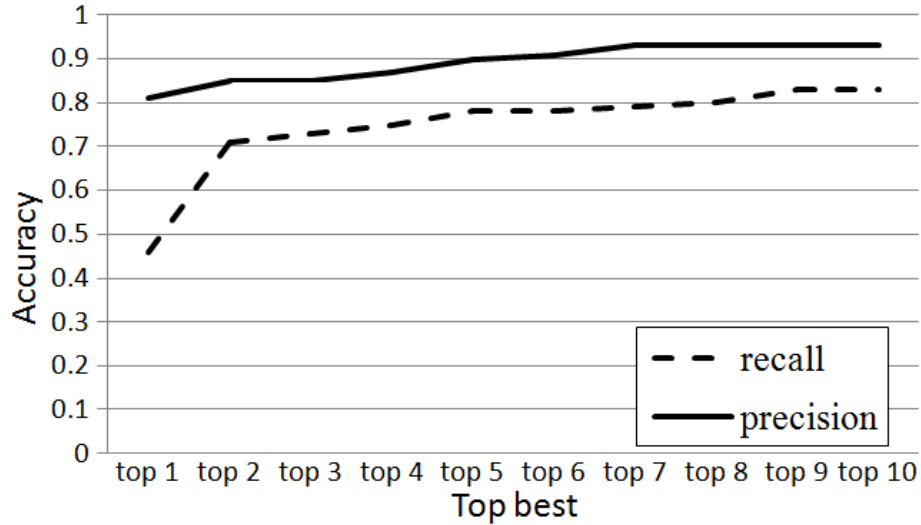


Figure 2.10. Graph showing results of precision and recall using the overlap metric for video understanding of top 10 functional unit results.

of these two overlaps was used to calculate the precision and recall and F-Score. Using FOON achieved higher F-Scores than not using FOON, as object and motion nodes in a video are perceived much better when using FOON as reference.

2.5.4 Task Inference (Recipe Classification)

We deployed our algorithm for recipe classification of unseen cooking videos. We used eight videos, including one salad recipe, two omelette recipes, two bread recipes, one cake recipe, one noodle recipe, and one sandwich recipe for the test. We classified all the recipes in FOON into 13 classes of recipes-Cake, Pizza, Bread, Omelette, Soup, Barbecue, Sandwich, Smoothies, Pasta, Coffee/Tea, Salad, Mashed potatoes, and Other.

Task inference was performed after all functional units in a video were identified. All the identified objects-in-action used in the video and the identified functional units equally contributed to the task inference stage. To classify a video to a recipe, clusters of recipes were created using all videos in the train set. The similarity distance between the current video and every (recipe) cluster was calculated, and the closest cluster was selected as the recipe associated with the video. To calculate the similarity distance between the current video and a cluster, the similarity of the video

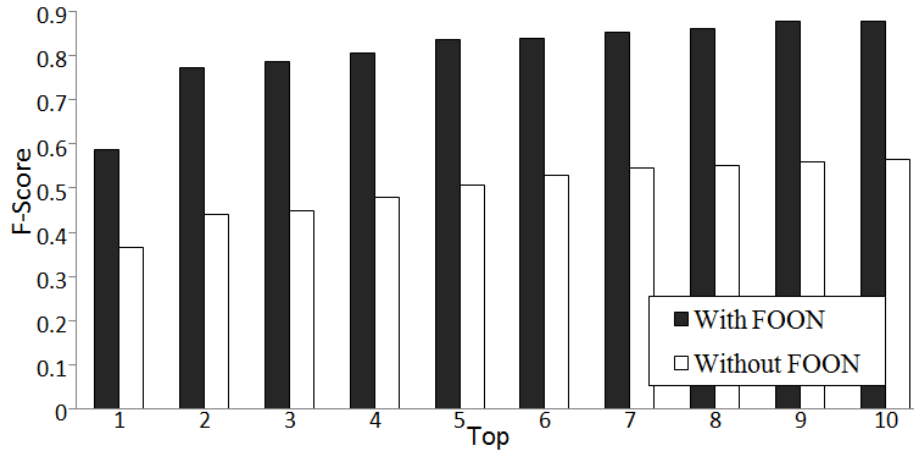


Figure 2.11. Graph showing calculated F-scores for video understanding with and without FOON.

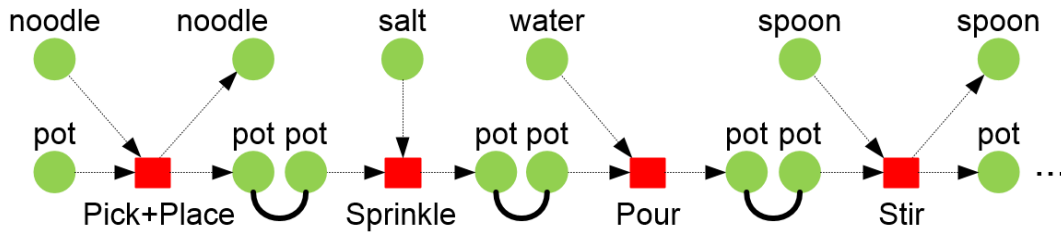


Figure 2.12. Illustration of identified functional units for noodle-cooking video (identified as noodle by proposed pipeline)

with each of the videos in the cluster was calculated and was averaged. The similarity distance between a video and a recipe was computed as the similarity of functional units in the video with the similarity of functional units in the recipe aggregated with the similarity of the used objects in the video with the similarity of the object nodes in the recipe. In the similarity comparison, we did not check the order of functional units. The recipe class with the highest similarity was assigned to the video. Figure 2.12 shows the identified functional units of a video demonstrating a cook making noodles.

The top 2 results of recipe classification are shown in Table 2.3. The recipe classification algorithm returned the predicted class names based on their confidence scores. If the class name with the highest confidence is equivalent to the ground truth class name, the classification is assumed as correct.

Table 2.3. Recipe classification results for manually and automatically labeled objects based on top 1 and top 2 results.

Used Procedure	Top 1	Top 2
Manually labeled Objects	37.5%	100%
Automatically labeled Objects	25%	75%

As shown in Table 2.3, the algorithm using FOON can approximately guess what recipe is being cooked in the video, assuming that all objects in the video sequence are identified correctly. The motion of the objects can also insinuate the type of recipe activity that is occurring.

2.5.5 Discussion

Several studies have worked on activity recognition using either knowledge bases or other methods, but they represent a video with a sentence or a label for the activity. Our work outputs sub-graphs representing short activities for each part of the video. As such, our work is incomparable to other work. We analyzed our work through the overlap metric and compared two approaches for video understanding-pipeline using FOON and pipeline not using FOON. It is clear that some methods in the literature can be substituted with the method we use to integrate with FOON, but our current focus is to prove that FOON is a powerful knowledge representation that can understand video and would be able to semi-automatically build itself in the future.

Although the proposed framework uses information from multiple views throughout the video, it applies information from only one camera at each specific time. However, due to the importance of simultaneous multi-view applications, we discuss briefly a few ways that the framework can be integrated into a multi-view system. The proposed framework can be applied individually to multiple videos in a multi-view system. Individual predictions can be gathered from multiple deployments of the framework. The predictions can be further combined to reach to a final prediction of the actions and activity in the video. We can also combine the proposed framework from a multiple view aspect at the confidence level. Confidences of objects can be extracted at each view and fused to reach a final confidence for the objects. The framework can further run as proposed.

The goal of the proposed framework is to identify the actions and tasks in a video. The framework can be used as the vision system of a robot chef or in any robotic system that deploys and manipulates utensils, such as a robot carpenter, robot waiter, etc.

2.6 Conclusion and Future Work

The main objective of this chapter was video understanding with the help of a knowledge representation. We use the FOON [78] as the knowledge representation for video understanding. We proposed a pipeline for video understanding using the functional object-oriented network (FOON) and deep neural networks and make use of low-level image features together with deep networks to identify objects of interest. Using objects of interest (objects-in-action) and deep motion understanding, we associate the actions in a video with the correct functional units in the knowledge representation (FOON). We demonstrated that using FOON significantly improves the performance of video understanding in comparison to not using FOON.

Our current pipeline is a big step towards automatically extending the knowledge representation graph, which presents a significant improvement to network applications, such as robots solving manipulation problems given a target goal. In future work, we would like to explore other methods of identifying objects-in-action, incorporate object recognition confidences to handle misidentified objects, utilize states of objects [122], and incorporate history of events for inference using FOON. We are also working on generalizing the knowledge contained within a FOON to achieve more generic video inferences from an expanded version of FOON [2].

Chapter 3: Understanding Cooking States

⁴ Robotic task planning and manipulation requires understanding both the objects and their state changes in the scene being manipulated. In the past chapter we analyzed object, motion and scenes for understanding videos in cooking settings. Many motions such as slicing implicitly contain information about the state change of an object. But, to explicitly utilize state modeling in understanding manipulation tasks we need to explicitly encode the state of a cooking object (e.g. sliced onion). Image understanding for object recognition and scene understanding have been very active topics in the last few years [25, 81, 30]. On the other hand, identifying object states has not captured much attention in computer vision and robotics research.

Objects' states can be defined as *characteristics into which the object could be transformed as a consequence of a human or robot activity*. A state can be observed and described as a form, texture, or color. For example, a tomato can have many states, such as sliced, diced, and whole. A whole tomato can be sliced and then diced in a sequence of cooking activity such as slicing and dicing. Assuming a robot chef wants to make a salad using a tomato, if it is provided with a whole tomato, it would need to wash it, slice it, and then dice it. If it is provided with a sliced tomato to begin with, it would need only to dice it. The intelligent robot chef would need to plan its motion differently based on the state of the provided tomato. Therefore, it would be necessary to not only recognize the object as a tomato, but also to identify the state the tomato is in. This is important for both fine-grained human activity understanding and robot task planning and manipulation control.

Robots also need to perform different manipulations or grasps to achieve different states of a planned task [123, 124, 125, 126]. Different states of an object or transiting an object from one state to another requires different types of grasping; for example, a whole carrot is grasped differently than a sliced or grated carrot [127, 128], or holding a whole carrot for slicing, holding a half carrot for

⁴This chapter was partially published in [1]. Permission is included in Appendix B.

grating, or holding a julienne-cut carrot for dicing each need unique types of grasping. Receiving on-line feedback from the environment would give the robot the sufficient knowledge required to decide on the type of grasp.

In this chapter, we provide a definition for object states, and provide a taxonomy of states for further analysis of cooking state changes. Using the taxonomy we introduce a dataset of cooking states for the public and propose novel models for joint object and state recognition. We also go one step further and parse the Recipe1M dataset for cooking states and create models to encode sets of states in a given image. We finally experimentally show that using state encoding can help improve ingredient set recognition from an image.

3.1 The State Identification Challenge

In this section, the state identification challenge and the dataset collected for the challenge are introduced, and the data collection procedure, dataset statistics, and details of the dataset are discussed.

3.1.1 The Challenge

In our daily lives, we perform tasks by paying attention to objects and their states and how they interact with each other. Like humans, one of the main tasks of an intelligent robot is to properly manipulate the environment. For a smart robotic system to perfectly manipulate the environment, it needs to acquire accurate knowledge of the environment, objects, and their affordances and status. An object can contain various shapes and states, therefore introducing various ways of manipulation. For example, when making an omelette, we need to dice peppers and onions. To dice a pepper, we need to grab the whole pepper, place it on a cutting board, use a knife to cut it in half, and, finally, cut it into julienne cuts and dice the cuts into small pieces. We can observe that the simple action of dicing a pepper requires knowledge of four different states for us or an intelligent robot. As humans, we get constant feedback from the objects (pepper). A robot needs to also gain feedback from the state of the pepper to decide how to continue the cutting. Knowing the current status of an object helps the robot with how it approaches the manipulation of the

object. In this example, for simplicity, we classified the states of an object into 4 different states, but in a real-world environment, the actual states of an object are continuously changing.

This example demonstrates the need for classification of an object (pepper) into a diverse set of states (whole, half, julienne, diced). Thus, we introduce the state identification challenge for the first time in this chapter. We define a state of an object (such as a tomato) as the various physical shapes (diced, paste, juice, or whole) into which the object can be transformed as a consequence of human or robot activity. We propose and anticipate that by solving the state identification challenge, we can step towards accurately understanding and executing robot manipulation tasks such as grasping. One of the main problems and applications of robotic systems is the cooking scope. In this study, we designed the state identification challenge for cooking objects. We analyzed cooking objects and their states by looking into the statistics extracted from the knowledge representation introduced in [78]. We discerned the most frequent objects in this knowledge representation and explored their states. State analysis shows that there are two major states for cooking objects—shape change and surface change. Hierarchical exploration shows that there are three main states under shape change, namely separated, morphed, and merged, each of which can further be represented by finer states. Also, surface change can be divided into two states, color and texture change, which, in turn, have finer representations themselves. Figure 1 depicts the hierarchical representation of explored states. In some cases, an object can have a combination of multiple states simultaneously. In this study, for simplicity, we assumed that an object can have only a single state at a time (in one snapshot).

A total of 22 fine states is shown in Figure 3.1. These states represent the whole state space that all important objects from [78] can span. We selected only 10 states (shown in gray in the figure) that are representative of the whole state space for our problem. The main reason to this state reduction is the lack of training image samples for the eliminated states. We can represent the state space in any other scope with a similar graph and further analyze the problem in that scope. In this study, we focused only on the cooking state space.

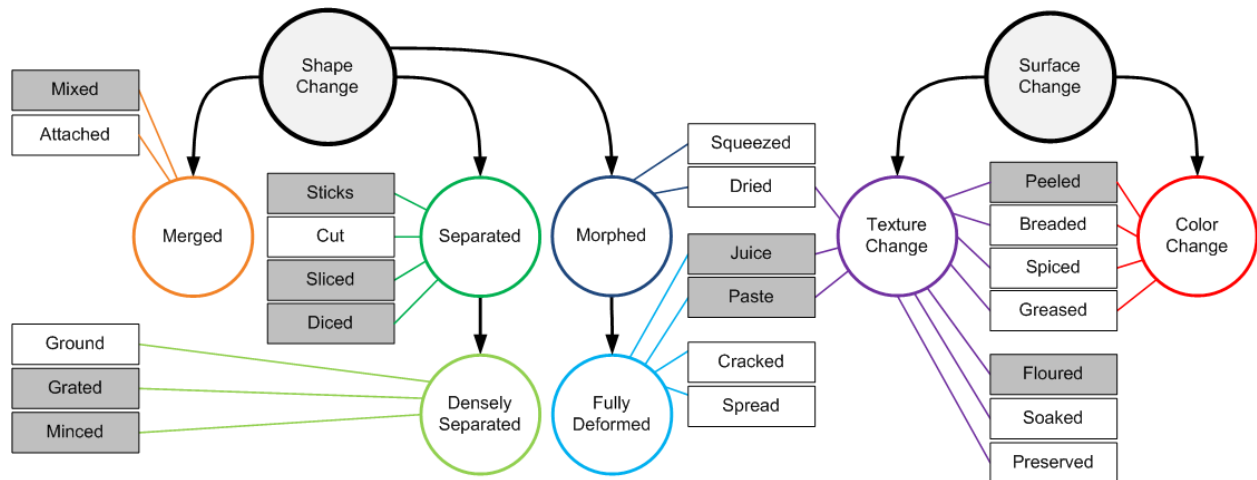


Figure 3.1. An illustration of the state taxonomy created by analyzing the FOON knowledge representation for the state identification problem.

3.1.2 The States Identification Dataset and Statistics

Dataset images were crawled through the Google search engine using a keyword combination of each object and state (such as "tomato" and "sliced"). The links to the crawled images were exported to a file, downloaded, and reviewed to remove unrelated images (such as cartoon images). Using the Vatic annotation tool, [29], images were published through a local server and dispersed to multiple workers for manual labeling. The labels were further reviewed and were gathered into a dataset of states. A small fraction of the dataset was labeled through the labelbox tool [1]. The collected dataset consisted of 17 main cooking objects, including tomato, onion, garlic, green pepper, potato, carrot, strawberry, egg, mushroom, bread, beef/pork, chicken/turkey, cheese, butter, dough and milk and 11 classes of states (whole, peeled, floured, sliced, diced, grated, julienne, juice, creamy, mixed, other). The total number of images in the dataset is 9309 - 6498(70%) training, 1413(15%) testing, and 1398(15%) validation images. The statistics of each class in the dataset is depicted in Figure 3.2. The classes "whole" and "sliced" contain more than 1000 images, and the other classes contain approximately 700 to 1000 images.

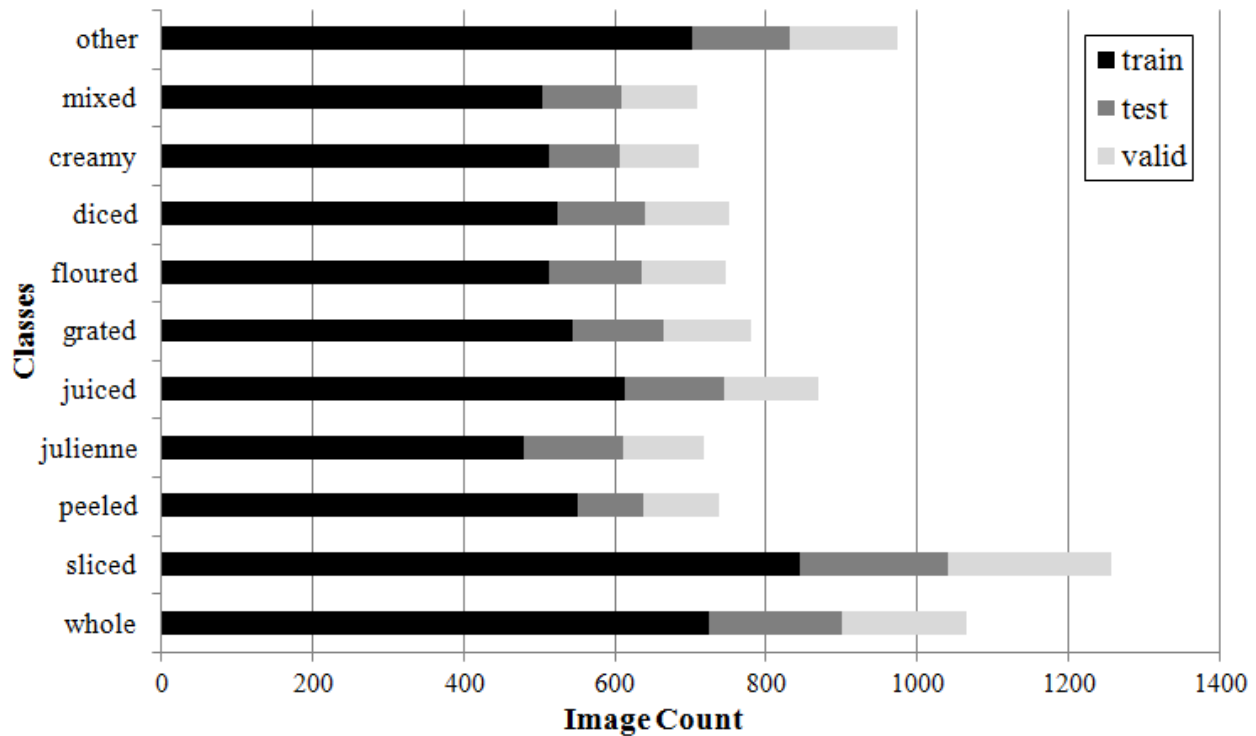


Figure 3.2. An illustration of the eleven state class statistics in the cooking dataset of over 10,000 images.

3.1.3 The Dataset Details

In this section, we give a concise definition of each of the 11 states to clearly state what each state represents. The whole state contains objects in their original format and shape, such as a whole pepper or a whole chicken, as shown in the first column of Figure 3.3. The peeled state contains objects that are peeled but not cut, sliced, or morphed, such as a whole peeled egg, onion, garlic, or tomato, (shown as in column 2 of Figure 3.3). The floured state as depicted in column 3 of Figure 3.3 contains objects that are floured. The grated state comprises of objects that are densely separated, such as bread crumbs, minced garlic, or grated egg (column 4 of Figure 3.3). The julienne state includes objects such as carrot sticks, French fries, julienne pepper, or shredded meat (column 5 of Figure 3.3).

The *diced* state contains diced or chopped objects, such as diced onion, tomato, and strawberry, chopped meat, and butter, and cheese cubes (Figure 3.3, column 6). The *sliced* class contains objects



Figure 3.3. Example images of eleven classes of the dataset. Each column of images in the figure represents a state from whole, peeled, floured, grated, julienne, diced, sliced, juiced, paste, mixed, other.

that are thinly-sliced such as sliced carrot, pepper, onion, tomato, meat or chicken slices, toast, and butter and cheese slices (Figure 3.3, column 7). Objects that are cut in other ways (such as cut in half or diced) are not considered sliced. The *juiced* class contains objects such as milk, melted butter, and tomato juice, (column 8 of Figure 3.3). The *creamy* state contains objects that are creamy, such as cream, creamy butter or cheese, garlic or tomato paste, and mashed potato (Figure 3.3, column 9). The *mixed* class contains a scramble of multiple objects such as salads (Figure 3.3, column 10). A final class called the *other* class is created that includes any state not listed in the previous states. A potato cut in half, squeezed lemon, images with multiple states, and an unmixed salad are in this class (Figure 3.3, column 11). Note that each object contains only a subset of the 11 states.

3.2 Baseline State Analysis

We analyze the proposed state dataset with a Resnet baseline model and provide results in this section. Like other recent image classification problems in the last few years, we propose to solve the problem with a deep convolutional Resnet baseline. Our model uses the Resnet base model up to the 46th activation layer [25] as its basis. We added a layer of 1x1 convolution, two layers of convolution, and a layer of global averaging before the 11 class soft-max layer. We used batch normalization in each layer for normalization and regularization purposes and added randomness

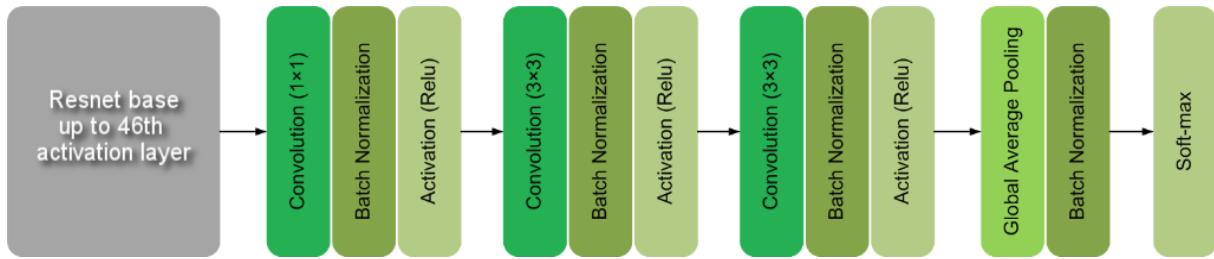


Figure 3.4. Proposed baseline network structure which includes a Resnet base [25] followed by three layers of convolution, batch-norm and relu.

to the network. The structure of the network is depicted in Figure 3.4. The 1x1 convolution was added to make the feature map set shallower. The convolutions were added to capture new spatial features for the specific state identification problem.

The network included more than 19 million parameters; therefore, the Resnet base, pre-trained on Imagenet, was used for training the state identification model. The pre-trained weights of the model initially were frozen in the first step and, further in the training procedure, the whole network (with 11 classes) was fine-tuned. More details about the transfer learning procedure are given in Section 3.2.1.1.

State identification has a strong correlation with the type of object. For instance, butter cannot be found as grated, julienne, or peeled and rarely can be seen as diced. On the other hand, cheese can be grated and lemon can be zested. We took this knowledge into consideration by fine-tuning 17 individual models for each object in the dataset. Each object has a different number of states; for example, garlic has 5 states including whole, paste, minced, peeled, and sliced, but carrot can have 7 states including julienne, diced, and juice. Therefore, the last layer of Figure 3.4 was removed and a new soft-max layer was added to the model. The number of images for each object in the dataset was limited; therefore, we initially trained the network on the whole dataset and fine-tune the modified network on a small number of object-specific images.

3.2.1 Experimental Baseline Analysis

We designed three experiments. In the the first, we trained and tested the deep architecture on the whole dataset. In the second, we showed that fine-tuning the model individually for each

Table 3.1. Baseline classification accuracy for the top 2 results on the state dataset and Imagenet subset

	Model	State Dataset		Imagenet Subset	
		Top 1	Top 2	Top 1	Top 2
1	Resnet-based Model	80.4%	91.5%	78.5%	89.6%
2	Voting	82%	92%	-	

object improves the accuracy for state identification. In the third, we tested our model on a sample of Imagenet images and provided state labels for a subset of their dataset [129].

3.2.1.1 State Identification

The model was trained and then evaluated on an unseen test set. We used the Adam optimizer with a learning rate of 0.001, beta1 of 0.9, and beta2 of 0.999, froze the Resnet base of the model, and trained only the layers we added to the network for 100 epochs. We then fine-tuned all layers of the model (including the Resnet base) for 250 epochs with a learning rate of 0.000005. On-line data augmentation, l2 regularization, and batch normalization was performed to reduce overfitting. The average class accuracy calculated for the trained and tested sets are 81.4% and 80.4%, respectively (Table 3.1). We trained 2 other models using a Resnet base and similar architectures. Using the validation set, a weighted voting was performed between these 3 models, and the best combination of weights was used for the final model. As shown in Table 3.1, after voting the state recognition accuracy rose to 82%.

For fine-tuning the model for each individual object, we perform a 4 stage training. In stage 1 and 2 all layers but the last are frozen. In stage 3 our additional layers are unfrozen and in stage 4 the whole model is unfrozen. Learning rates for stages 1 to 4 are 0.01, 0.001, 0.00001, and 0.000005 respectively. Epochs for stages 1 to 4 are 40, 80, 120, and 160 respectively. The First 5 columns of Table 3.2 shows the classification accuracies of the fine-tuned model for each object. The object dough was removed from this set of experiments.

Table 3.2. Baseline state classification accuracy based on individual ingredient fine-tuning and the number of states per ingredient

Object	Top 1	Voting	States	Test Set
mushroom	95.6%	97.8%	3	45
onion	80.2%	85%	7	86
strawberry	92.6%	92%	4	68
bread	78.9%	78.9%	6	123
butter	69.7%	72.7%	5	66
carrot	78.5%	84.9%	8	135
egg	90.6%	89.2%	5	85
garlic	86.7%	85.3%	5	75
lemon	90.7%	94.9%	6	108
milk	100%	100%	2	40
pepper	96.1%	97.5%	5	76
potato	84%	88.3%	8	106
tomato	88.5%	91.1%	7	113
cheese	82.7%	78.7%	4	75
beef/pork	86.7%	86.7%	5	60
chicken	88.8%	89.7%	6	116
average	86.9%	88.3%	5.4	86.1

3.2.1.2 ImageNet Test

In this experiment, we contributed to the Imagenet dataset by providing state labels for them. For each object category in our dataset, excluding beef and chicken, 50 images were randomly selected from the Imagenet dataset. Beef and chicken were excluded because Imagenet does not contain cooking-related images for these two categories. The Salad synset was included for the experiments because it is considered a frequent image in cooking videos, thus leading to a total of 16 object categories and 800 images. The images were labeled with the 11 classes in our dataset.

Moreover, the trained model was run on the Imagenet subset and the average state identification accuracy on the Imagenet subset was reported as 78.5%. Individual accuracies for the top 1, 2 and 3 are shown in the last three columns of Table 3.3. In addition to the evaluation, we ran our model on all images associated to the 16 object categories and gave their labels. Then we manually checked the labels through an interface and keep the correct labels and discard the others. The state labels of the images and the dataset will be released on our website for download after the double-blind peer review process.

Table 3.3. Baseline state classification accuracy for individual ingredient in the Imagenet subset based on top 1 to top 3 results

Object	Top 1	Top 2	Top 3
mushroom	74%	84%	96%
onion	86%	94%	96%
strawberry	74%	86%	90%
bread	80%	96%	98%
butter	70%	88%	94%
carrot	96%	100%	100%
egg	62%	78%	86%
garlic	84%	90%	92%
orange	90%	96%	98%
milk	72%	80%	88%
pepper	78%	88%	94%
potato	72%	94%	96%
tomato	74%	82%	92%
cheese	84%	90%	98%
salad	90%	98%	98%
dough	70%	90%	96%
average	78.5%	89.6%	94.5%

3.2.1.3 State Analysis

Classification accuracy of all classes apart from the *other* class is at least 70% for the experiment on our test set (confusion matrix depicted in Figure 3.5.a). The other class includes various kinds of images, such as images of meals, sandwiches, images including a combination of various states (classes), and etc. The variety in the other class is the reason of such a low accuracy. We anticipate performing a joint detection and recognition procedure of states would improve the accuracy of classification in all classes.

A majority of the mistakes made by the model were on account of ambiguous and multi-state images as depicted in Figure 3.6. This also suggests that detection of all states inside an image rather than looking at the entire image as a whole may improve the state identification accuracy. Moreover, tracking the state of an object and assigning values for representing the quality of an object being in a specific state may improve state identification results.

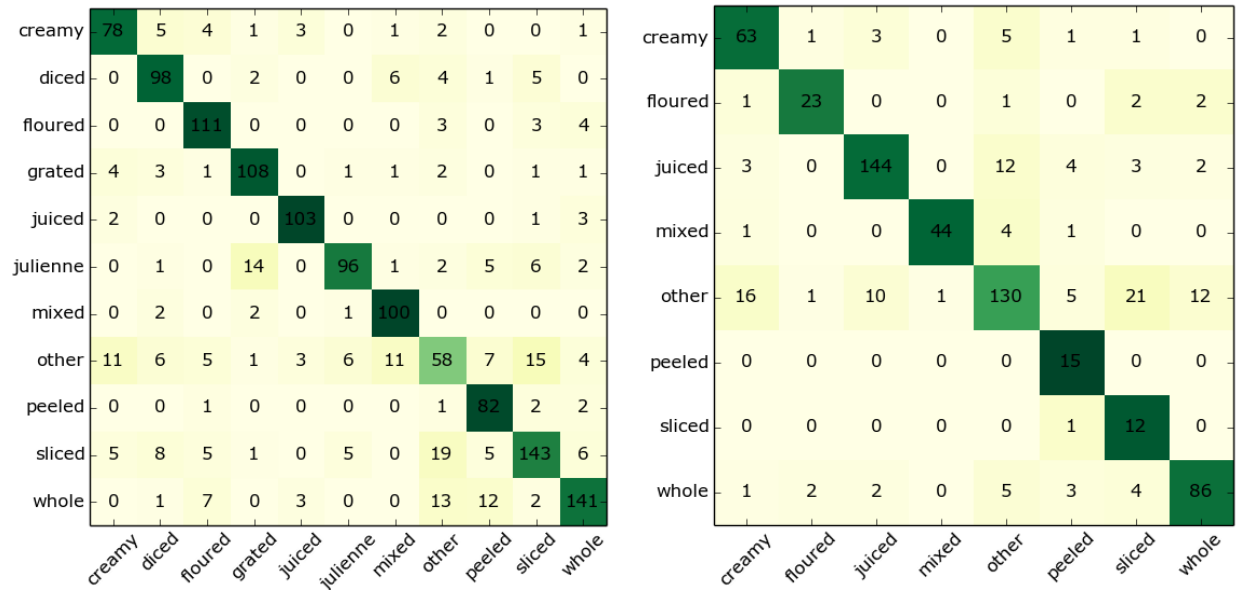


Figure 3.5. Confusion Matrices: a) Confusion Matrix of the Baseline on the State Identification Dataset. b) Confusion Matrix of the states baseline model on the Imagenet Subset

3.2.2 Imagenet Analysis

Unexpectedly, the average state identification accuracy on the Imagenet subset, despite having more ambiguous images, was only slightly lower than the classification accuracy on our dataset. Figure 3.7 shows examples of ambiguous images from the Imagenet subset. These images either contain multiple states such as Figure 3.7.c. or are out of our dataset scope such as Figure 3.7.a. Interestingly, the half peeled pepper, in Figure 3.7.d., was predicted as peeled although no image of a peeled pepper is included in our dataset. Figure 3.7.b. was counted as a wrong prediction, although the model’s first three predictions is whole, julienne and other. This example shows that the model is able to capture sufficient features, but does not have the tool to identify multiple states in an image simultaneously. The confusion matrix on the Imagenet subset is depicted in Figure 3.5.b without classes diced, grated and julienne due to the shortage in image samples.

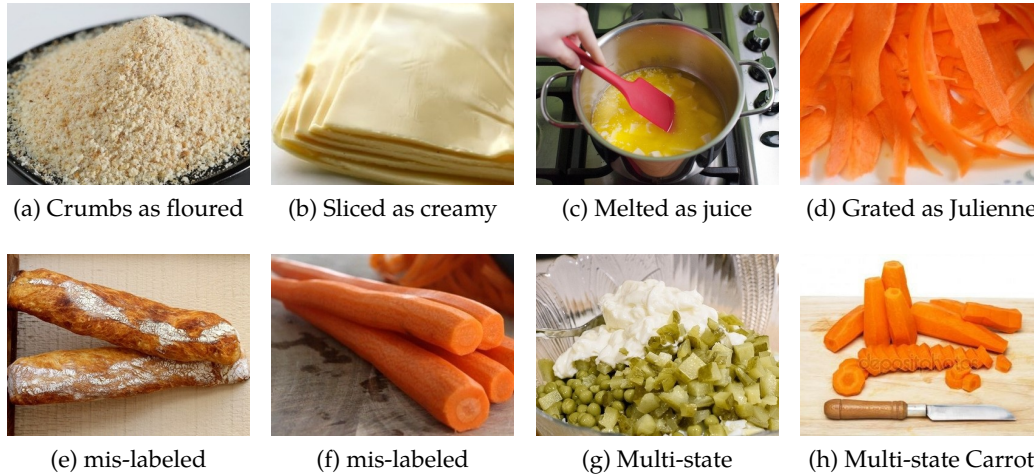


Figure 3.6. Samples of mis-predicted (a, b), ambiguous (c, d), mis-labeled (e, f) and multi-state (g, h) images. (e) is floured and (f) is peeled but they are mis-labeled as whole in the dataset.

3.3 Joint Object and State Recognition

States and objects are intertwined together, meaning that object information can help recognize the state of an image and vice versa. In this sections, we address the state identification problem in cooking related images and use state and object predictions together to improve the classification accuracy of objects and their states from a single image. We propose a pipeline for joint state and object identification. The pipeline includes a convolutional neural network, a language model and two MLP networks as shown in Figure 3.8. We apply a selector gate on the pipeline outputs to improve results as depicted in Figure 3.9.

3.3.1 Stage 1: Double Loss Convolutional Network

In the first stage of the pipeline, we use the Resnet architecture with two outputs- one for state and one for object classification. The two applications use the same weights apart from the last layer. The loss applied for object and state classification are defined separately and trained simultaneously. The network outputs two different sets of confidences via the soft-max layer, one for the state classes, $[P(state_i)_{i=1:N_{states}}]$, and another for the object classes, $[P(object_i)_{i=1:N_{objects}}]$, as shown in Figure 3.8. The notations N_{states} and $N_{objects}$ are the number of states and objects

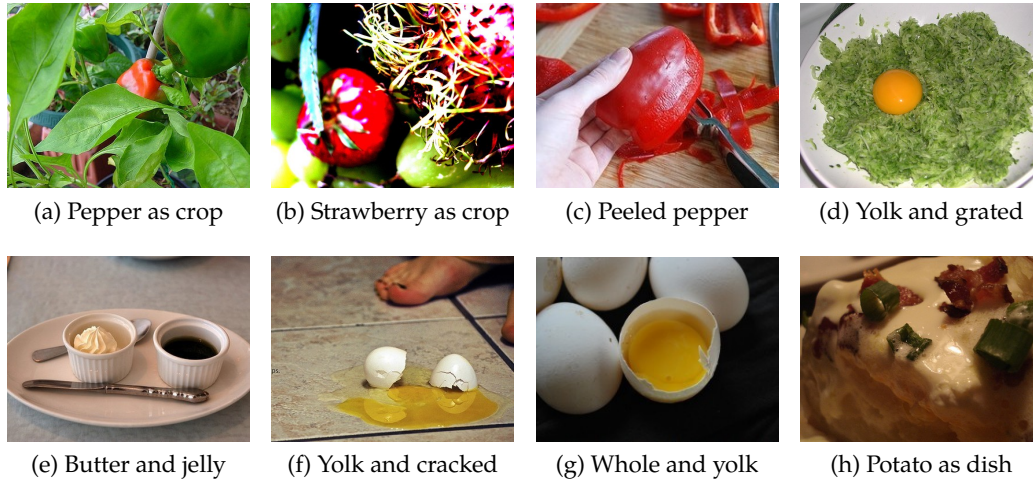


Figure 3.7. Samples of multi-state or ambiguous images in the Imagenet dataset where the model may confuse the classes

respectively. The soft-max confidences are the first set of probabilities we obtain for object and state classification. We name them as prior probabilities of each object (or state) occurring in the image.

3.3.2 Stage 2: Language Knowledge Based Features

In natural language processing, documents, sentences, and words are processed to extract meanings, relationships and word embeddings. For our analysis in this work, we use the *Concept-Net* knowledge representation, which is more powerful than the widely used Word2vec [130], and the *Google N-gram Viewer* to quantify word relations.

Concept-Net is a language knowledge graph that includes words and phrases as nodes and natural language relationships between the nodes as edges [131]. Concept-Net defines and implements a class of language- and source-independent relations between words and phrases including *IsA*, *UsedFor*, and *CapableOf* and also associates weights with every relationship. Weights of relations are calculated based on an aggregation of weights from various sources. We use the weights from the *RelatedTo* relation (or assertion) of the Conceptnet API to quantify the relationship of a specific state (e.g. sliced) with a specific object (e.g. bread).

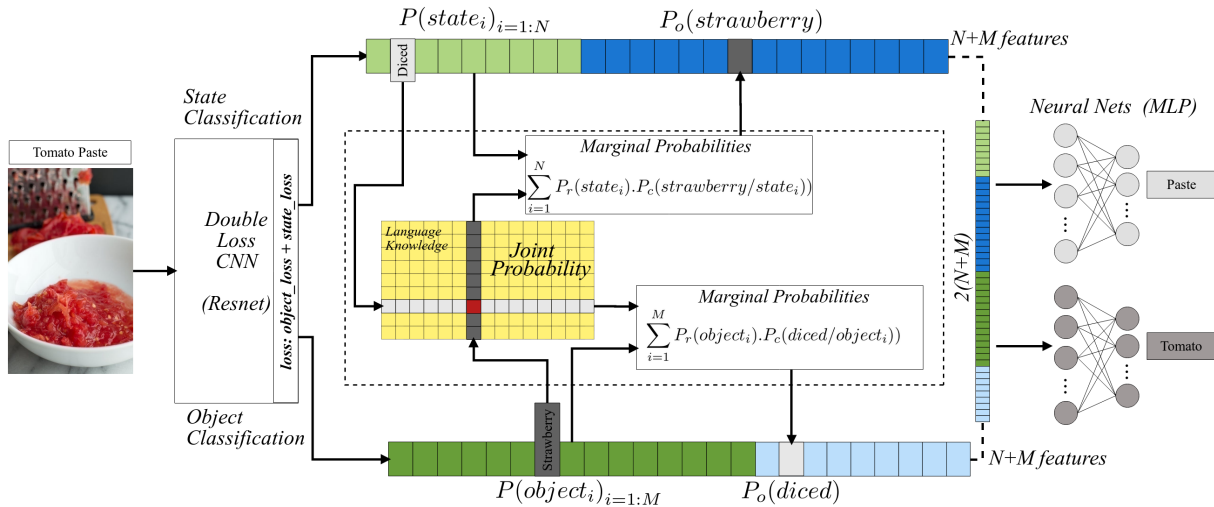


Figure 3.8. Pipeline for simultaneous state and object classification using language knowledge and post marginal probabilities for states and objects.

In natural language processing, an N-gram is a sequence of N items (e.g. words) in a bed of various documents called a corpus [132]. The frequency of two or multiple words happening together (N-grams), can be representative of how related they are. The Google N-gram Viewer is a Google based search engine that shows the frequency of any N words occurring consecutively in Google’s text sources [133]. We use the frequencies extracted from the Google N-gram Viewer to represent the relationship between states and objects.

3.3.2.1 Feature Extraction

The correct identification of objects is associated with the correct identification of states and vice versa. We use the Concept-Net and the Google N-gram Viewer to quantify the relationship between the states and objects in the dataset. We first define a set of words associated with each object and a set of words associated with each state. For instance, for the object *potato* we define the set $\{potato, potatoes\}$ and we define $\{creamy, paste, mashed, mash, softened, whipped\}$ as the set representing the state *creamy*. To calculate the joint probability of an object (e.g. potato) and a state (e.g. creamy), every pair of object and state from the two sets is looked up in Concept-Net or the N-gram Google Viewer to derive a relatedness value. The maximum and the mean values for each pair are recorded (e.g. potato-creamy). The confidences are normalized so that the sum

of all probabilities of a state over various objects and the sum of all probabilities of an object with different states each sum up to 1.

We calculate the marginal probabilities for each object assuming the state prior probabilities and joint (conditional) probabilities ($[P(object/state_i)_{i=1:N_{states}}]$) derived from the language knowledge source (e.g. Concept-net or Google N-gram Viewer). We conversely compute the marginal probabilities for the states using joint (conditional) probabilities ($[P(state/object_i)_{i=1:N_{objects}}]$). The relations for marginal probabilities for each object $P(object)$, and state, $P(state)$, is given in (3.1), and (3.2) respectively.

$$P_o(object_j) = \sum_{i=1}^{N_{states}} P_r(state_i).P_c(object_j/state_i) \quad (3.1)$$

$$P_o(state_j) = \sum_{i=1}^{N_{objects}} P_r(object_i).P_c(state_j/object_i) \quad (3.2)$$

In (3.1), and (3.2), P_c is the conditional probability of an object in respect to a state or vice versa which is derived from the language knowledge, P_r is the output confidence from the Resnet, and P_o is the marginal probability.

3.3.3 Stage 3: Neural Network Predictions

The marginal and prior probabilities are concatenated together to create a feature vector of size $2 \times N_{objects}$ and $2 \times N_{states}$ for objects and states respectively. The concatenated object and state features are merged together to create a final feature vector with size $V_{final} = 2 \times (N_{states} + N_{objects})$. The feature vector V_{final} is given as input to two separate MLP networks for object and state classification respectively as shown in Figure 3.8. A three layer MLP is selected using the validation set as the finalized architecture of the networks.

3.3.4 Stage 4: Model Refinement

The pipeline converts correct predictions into incorrect predictions in some cases. To reduce these conversions, a refinement procedure is proposed that starts training after the double loss CNN has finished training. The refinement model is trained to predict the probability of an image

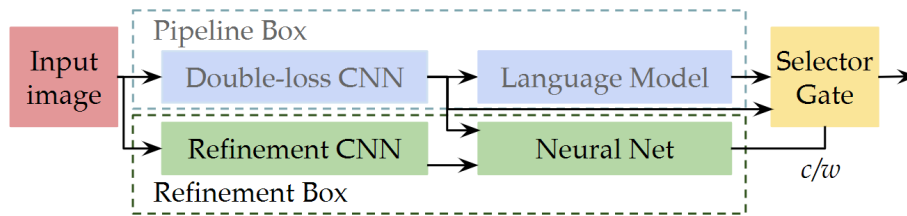


Figure 3.9. Illustration of the pipeline refinement module that corrects (refines) the results if the initial model contains incorrect results.

being classified correctly by the pipeline. The refinement model contains a Resnet-based CNN which returns two outputs (classes) for a given input image; one output represents an image being classified correctly and the other represents the image being classified incorrectly. The two outputs are associated with two confidences. The two confidences are concatenated with the output probabilities from the double loss CNN from the pipeline. Two separate neural networks are trained for correct/incorrect object (and state) probability predictions using the concatenated feature vectors. The outputs from the MLP are used as a selector for a gate selector block. A value of one for the selector output, means that the initial prediction is correct and the object (or state) confidences from the double loss CNN are used for predictions. A value of zero for the selector output, means that the the probabilities after language knowledge incorporation should be used for predictions. The refinement model is depicted in Figure 3.9.

3.3.5 Results and Analysis

We implemented the Resnet model in Tensorflow and initialized with pre-trained weights from Imagenet. The single classifier layer was removed and a double classifier layer was added for states and objects. We trained the model for 15 iterations and with an initial learning rate of 0.01. Only weights from the last block of Resnet were trained and the rest were kept frozen. The relatedness values of objects and states were downloaded from the Concept-Net (or Google N-gram Viewer) Web APIs using the Python Request library and the normalized versions of the relatedness values were recorded as joint probabilities. The final features were then computed and then given to MLPs as mentioned in Section 3.3.3.

Table 3.4. States and object classification accuracy on the test set with and without using Concept-Net (Concept-Net as CN, Google N-gram as GN).

Model	States	Objects
Resnet	79.4%	74.1%
(Resnet,CN) + SVM	79.7%	74.2%
(Resnet,GN) + MLP	80.1%	74.2%
(Resnet,CN) + MLP	80.4%	74.3%
(Resnet,CN) + MLP + Refinement	80.9%	75%

We compared the pipeline with other methods and report the results in Table 3.7. We compared the pipeline with the raw initial confidences, the linear combination of the initial confidence and the marginal probabilities from Concept-Net, and an SVM-based version of the pipeline. The results show that all methods containing a language knowledge outperform the Resnet network as shown in Table 3.7. The neural network based method that uses features from the Resnet output and the Concept-Net features outperforms all other methods. Results in Table 3.7 show that self-correction using the refinement model improves the results even further.

Figure 3.10 shows an instance of an incorrect result (diced strawberry) converting to a correct result (tomato paste) when using Concept-Net. Concept-Net can make mistakes. For example grated butter has a high relatedness confidence in the Concept-Net graph although in the real world it is unlikely to see grated butter often. Therefore, it is easy to flip a correct *creamy butter* to an incorrect *grated butter*. The refinement model has the ability to prevent some of these cases.

3.4 Generating Multiple States and Ingredients

Assuming that an image contains only one ingredient and state is a distant assumption. In this section, we assume that we have an image of a meal that may contain multiple ingredients and states. Therefore, we explore identifying multiple states and ingredients from an image (Figure 3.11). Predicting ingredients given a single image of a meal is challenging. In this problem setting, correctly ingredient prediction would only depend on the global (visual) features of the image. As previously discussed and experimented in [1], ingredients and their states are deeply associated and complementary concepts to one another. We argue that adding a prior step to visually discern

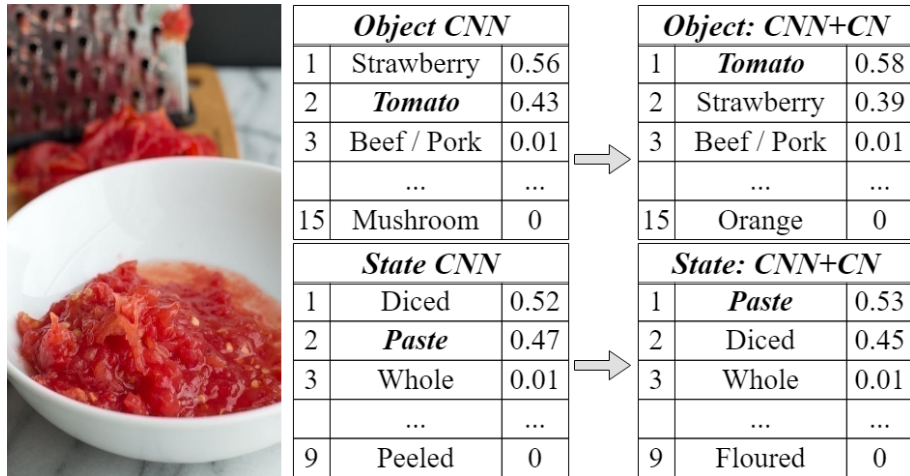


Figure 3.10. Objects and states CNN probabilities vs Concept-Net (CN) probabilities. Probability of diced strawberry is lower than tomato paste when CN is used.

the states (of ingredients and the meal) would enhance the performance of the ingredient prediction model. We therefore exploit the associations between ingredients and their states and how they are connected with categories of recipes. We use the Recipe1M dataset [99] in our experiments. But, this dataset does not include explicit states labels. In this section, we first explain how we define states in general and for the ingredients in the Recipe1M dataset. Afterwards, the model proposed for ingredient prediction given a single image and a set of states is described.

3.4.1 State and Token Embeddings

3.4.1.1 States Exploration

Recipe and ingredient detection from a given image has been the study of many research in the past few years [99, 83]. On the other hand, states, has not been recognized as a popular research area [1, 101]. We explore ingredient states (e.g. sliced tomato) and analyze its effect on ingredients detection. In [101], states of ingredients are categorized into different state (e.g. mixed, grated, peeled) categories (classes). We use these categories as the core set of state classes and semi-automatically derive and extend state classes from the Recipe1M dataset in 3 stages:

- *Core set*: A set of state classes is defined as the core set inspired by the states defined in [101].



Figure 3.11. A picture of carrot ginger soup and its associated ingredients (e.g. onion) and their states (e.g. chopped).

Table 3.5. Ingredient phrases from Recipe1M and the ingredients and their states extracted from the Recipe1M dataset.

Ingredient Description	Ingredient	State
1/2 cup green onions, chopped	onion	chopped
1 teaspoon ground cinnamon	cinnamon	ground
1 cup white mushrooms, sliced	mushroom	sliced

- *Alias extraction:* Aliases of the states in the core set are manually defined. The Recipe1M dataset is also semi-automatically parsed for other aliases.
- *Manual addition:* The past tense form of verbs and highly frequent adjectives are extracted from only the *ingredient descriptions* of the Recipe1M dataset. If similar to any of the core states, they are added as an alias. Otherwise, if the frequency of the derived state is higher than a threshold, a new state class is added.

Table 3.5 shows a few instances of ingredient descriptions and their associated states. Figure 3.12 depicts the states derived from the Recipe1M dataset and their frequency.

To perform experiments without the bias introduced by the state list suggested in [101], we disregard that list and analyze all adjectives and verbs with one dominant meaning from the Recipe1M dataset. Similar to the process of extracting states, these adjectives and verbs are clustered into a set of token (e.g. using aliases) classes. Also, less frequent words were put into

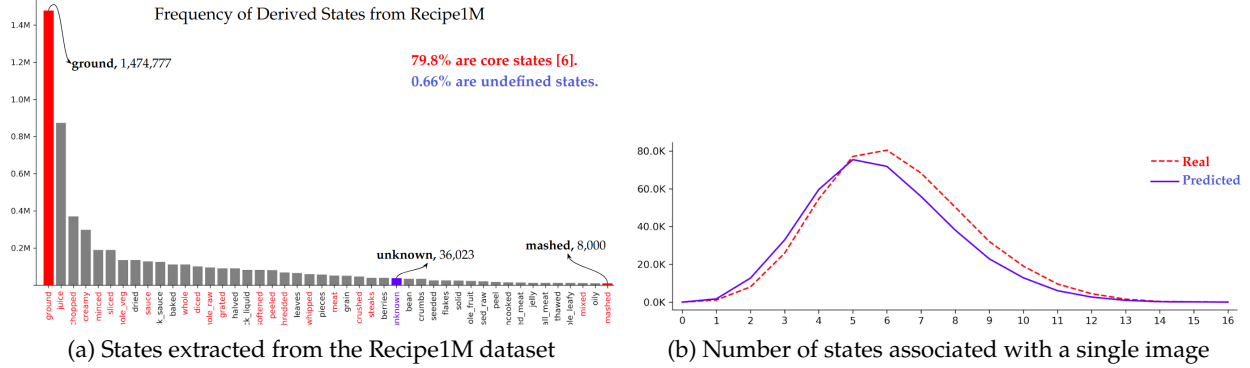


Figure 3.12. Histogram of states extracted from the Recipe1M dataset and number of states associated with a single image.

a separate unknown class. In both of the above processes, words without a visual effect on the image (e.g. salted, estimated, learned, considered) are removed from the classes.

3.4.1.2 Embeddings

Embeddings for states and words are derived from every given image. Inspired by the transformer decoder presented in [35, 134], we design a model to extract state (or word) embeddings. In this sub-section the process of extracting the embeddings is explained. Given an image I , feature maps $G \in R^{C \times S \times S}$ are computed from a Resnet encoder. G is then flattened to $F \in R^{K \times D}$ and fed into the transformer decoder from [134, 35]. At each step (t) of the transformer decoder, going through a softmax layer, a state logits vector $s_t \in R^{N \times 1}$ is predicted, in which N is the number of state (word) classes. To model dependencies between states, predicted states are fed back into the Transformer decoder, to predict future states. The predicted states are concatenated into a matrix $S \in R^{N \times L}$ where L is the maximum number of possible states included in a single image. Max-pooling is applied to remove order of predicted states [35]. Applying max-pooling to the matrix S , produces a vector of probabilities $P_s \in R^{N \times 1}$. The positional encoding is also removed from transformer decoder to eliminate any order encoding. The vector P_s further is encoded through a fully connected layer into a state embedding vector $E_s \in R^{K \times 1}$. Equation 3.3 shows the operations applied to extract the state embedding.

$$E_s = f_p([\mathit{f}_D(F)]_{i \in \{0, \dots, k\}}) \quad (3.3)$$

In which k is the number of image features, f_D is a decoder (transformer), $f_D(F)_i$ is the i -th output of the transformer and f_p is a max-pooling operation over all the transformer outputs, and $[]$ is the concatenation operator, concatenating all the transformer outputs. The state embedding vector contains values between 0 and 1 that associate the input image to each of the possible states. The same steps are applied to the input image to compute a word probability vector $P_w \in R^{M \times 1}$ and eventually a word embedding vector $E_w \in R^{K \times 1}$ of the same size of the state embedding vector. The state embedding and the word embedding together with the given image are used as encoded features to predict ingredients. In training the embedding models we use three loss terms: binary cross entropy loss, end token loss, and cardinality loss [35].

3.4.2 Ingredients Given States

The goal of this section is to exploit the associations of states with the process of predicting ingredients. Therefore we propose a model where given a single input image (cooked meal) I , and the state and word embeddings (E_s, E_w) , a list of ingredients associated with the given image (recipe) are predicted. In the first stage, state and token embeddings from the previous stage are concatenated with the image embedding as shown in Equation 3.4.

$$Z = [F, E_s, E_w] \quad (3.4)$$

where $E_s \in R^{K \times 1}$ and $E_w \in R^{K \times 1}$ are the two state and token embeddings, and E_I is set of flattened feature maps from the Resnet encoder, $F \in R^{K \times D}$. $Z \in R^{K \times (D+N_e)}$ is the concatenated features. The encoded feature matrix Z is fed as input to the transformer decoder. N_e is the number of state or word embedding vectors (i.e. 2). The transformer decoder is deployed from the well known transformer model proposed in [134]. Therefore, multiple streams of dot-product attention are applied to the embedding as in Equation 3.5.

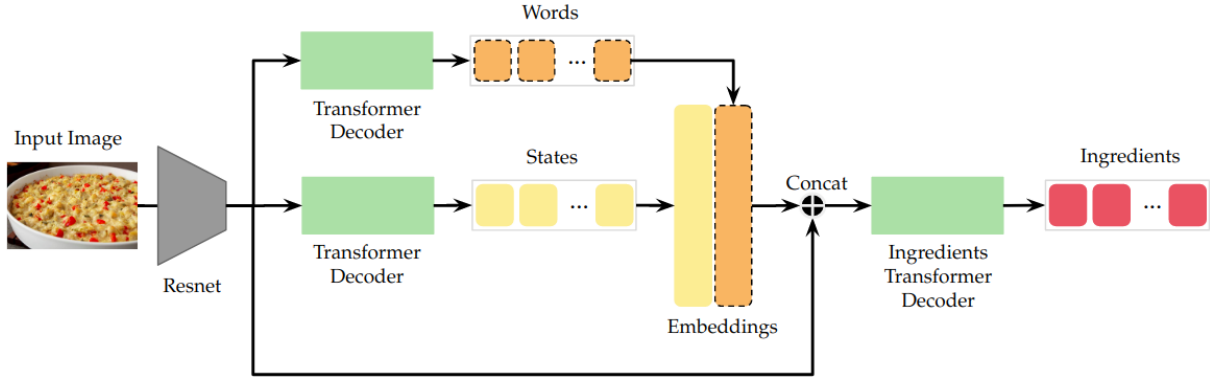


Figure 3.13. Two-step model that predicts ingredients from a single image. In the first step, given a cooking image two sets of state and word embeddings are created. In the second step, using the state embedding and the image embedding, ingredients are predicted one step at a time using a transformer decoder.

$$Z_{att} = \text{softmax}\left(\frac{ZQ^T}{\sqrt{d_q}}\right)Z \quad (3.5)$$

where Q is the query matrix comprising of the previously predicted ingredient embeddings and d_q is the dimension of each ingredient embedding. All transformer operations, including multiple stacked attention layers with layer normalization and residual connections are applied, followed by a point-wise feed forward layer. The ingredient predictions are further passed through a max pooling layer [35] over time. A binary cross entropy loss is trained over this output to model the ingredients in an order-less manner. The proposed transformer decoder is trained step by step as in [35]. Figure 3.13 shows an abstract illustration of the model.

Similar to the state (and word) embedding models proposed in Section 3.4.1.2, ingredients are predicted one step at a time through a transformer decoder while feeding the previously predicted ingredients into the model. The training of the transformer decoder is also executed one step at a time by feeding previous ingredient predictions as word embeddings to the model. Similar to the states (tokens) embedding training, we use three loss terms: binary cross entropy loss, end token loss, and cardinality loss [35] for training the ingredient prediction model.

The state embeddings fed to the model are the output of the pooling layer from the transformer decoder. We do not use one-hot encoding of the states (or tokens) for two reasons. The one-hot encoding has to be fed one by one to the model and learn its own embedding which makes the

model more complicated and harder to learn. Secondly, the output of the pooling layer has richer context, including information from all states at all steps, even after the end token, thus letting the model choose where and what to attend at.

3.4.3 States Generation Analysis, Experiments and Results

3.4.3.1 Dataset

We used the well known Recipe1m dataset for our experiments [99, 135]. This dataset contains over 1 million recipes. The majority of the recipes in this dataset are associated with an image. We cluster the ingredients into the same ingredient clusters proposed in [35]. They cluster similar ingredients into ingredient categories such as oil (contains olive oil, vegetable oil, etc), cheese (including 400 types of cheese), and pepper (300 types of pepper). The actual number of unique ingredients in the dataset is 16823 which after clustering reduces down to 1488. We use these 1488 processed ingredients in our experiments.

Although the dataset contains descriptions for ingredients (e.g. 1/2 cup green onions, chopped), it originally does not contain any explicit states (e.g. chopped) for ingredients (e.g. onion). As suggested in Section 3.4.1.1, the ingredients descriptions are parsed and explicit states are assigned to ingredients of a recipe in a semi-automatic manner. After this process, which includes alias definitions and state clustering we define a total of 59 global states for all of the ingredients in the dataset. If no states are available in the ingredient description we associate a default state to it (e.g. oil is associated with liquid). Also, as suggested in Section 3.4.1.1, a set of more frequent words (verbs and adjectives) derived from ingredient descriptions are also assigned to each ingredient, leading to a total of 119 global tokens for all of the ingredients in the dataset. We also generate a third set of tokens in which the clustering is performed more loosely and very high frequent words with non-visual effects () are included. The third set of tokens, includes 152 tokens. The number of tokens in the states set (59), and second (119) and third (152) token set is shown in the last column of Table 3.6.

3.4.3.2 Implementation Details

We used the implementation from [35] in PyTorch4 and built on top of it. Our implementation of the transformer decoder for states extraction, state embeddings, and ingredient prediction were all done in PyTorch4 and incorporate the same settings as suggested in [35], which include 4 blocks and 2 multi-head attentions, each one with dimensionality of 256. As in [35] and many computer vision literature, we resize images to 256 pixels on their shortest dimension. We then take random crops of 224×224 for training. As the image encoder, the last convolutional layer of ResNet-50 model is used. The dimensions of the feature maps is of size $512 \times 7 \times 7$, which flattened is converted to size 512×49 . A maximum of 20 ingredients and 20 states (or tokens) are associated with each recipe. For recipes with more number of tokens, the remaining tokens are randomly excluded. The models are trained with the Adam optimizer until early-stopping criteria is met. We use IoU (Intersection over Union), and F1 for evaluation of the results. We assign weights 100, 1, 1 and 100, 1, 1 to the 3 loss terms for the ingredient prediction, and token embedding models respectively. For the states embedding model 20, 1, and 1 are assigned as loss weights.

We have experimented two sets of results: states embeddings and prediction, and ingredient prediction given a single input image. We explore each of these experiments.

3.4.3.3 Experiments on State Embeddings and Prediction

As for the states (tokens) prediction, we report IoU and F1 scores in comparison with the ground-truth list of states (or tokens) for each given image. As shown in Table 3.6, the IoU for the prediction of the 59 states and 119 tokens is 38.69 and 38.02 respectively. The IoU for the 152 tokens is lower, 32.25%, which may in one part be associated with the non-visual tokens added to the token classes. Table 3.6 shows F1 scores besides the IoU results for all three models: states encoder, tokens encoder, and the tokens+HF encoder. This kind of experiment has never been conducted on states of a single image, therefore we have not compared these results with other work. In [1], given a single image and a set of 9 states and 15 ingredients approximately 81% and 75% state and ingredient classification was reported. In comparison to that work, this model has two significant advantages. One, is that it does not assume a single state associated with the image, which is the

Table 3.6. States prediction for the (59) states encoder, the (119) tokens encoder, and the (152) tokens+HF encoder.

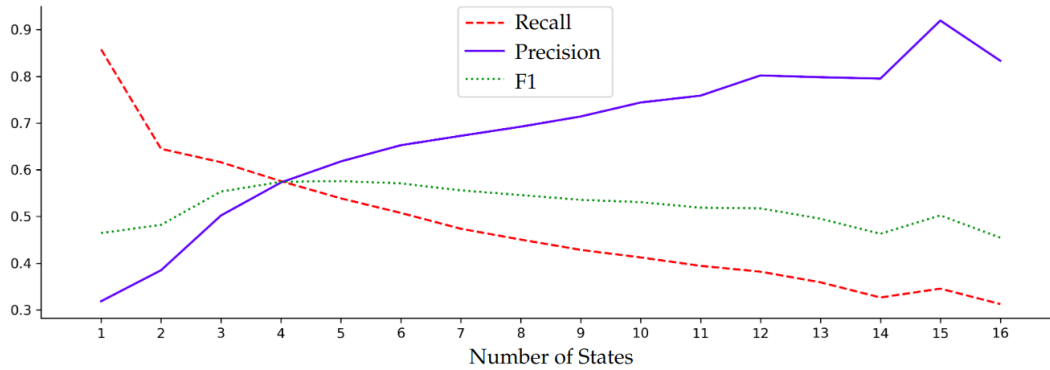
States Model	IoU	F1	count
States encoder	38.69%	55.8%	59
Tokens encoder	38.02%	55.1%	119
Tokens+HF encoder	32.35%	48.88%	152

case in most cooking images online. Second, it considers a much broader set of ingredients and their states. Therefore, this model is a much more realistic and robust model in comparison to a simple joint ingredient and state classification model [1].

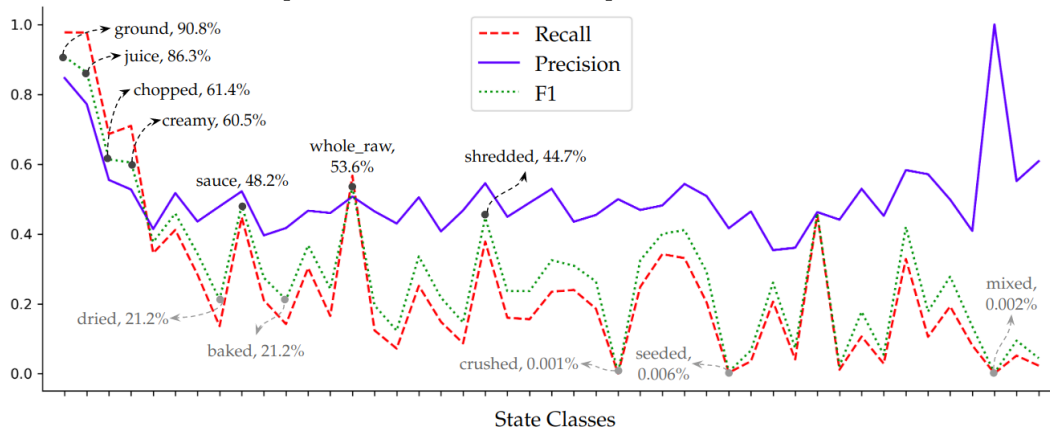
We have also analyzed the recall, precision, and F1 scores for different recipes based on the number of states included in a recipe (Figure 3.14.a). As depicted in Figure 3.14.a, recall decreases when the number of states increases, while precision increases when the number of states increases. This indicates that when the number of states increases, the predicted states are more accurate although the model is not able to predict many of the ground-truth states. The highest F1 score is achieved when the number of states is 4. The model performs better for some states and worse for others. Figure 3.14.b shows the F1 score for different state predictions excluding the states with lower occurrences. States are ordered in the order of their frequency (as in Figure 3.12). Precision for different states is more stable than recall, showing their accuracy of guesses by the model. The recall on the other hand degrades when the frequency of states reduces, which may suggest that having a more inclusive dataset would be beneficial. The model performs the best on the four states *ground*, *juice*, *creamy*, and *chopped* which are also very high frequent and originate from the core set of states [101]. The state *baked* although is high frequent performs unsatisfactory due to the various concepts it includes (i.e *cooked*, *grilled*, *baked*, *simmered*, *fried*, etc. The three states *mixed*, *crushed*, and *seeded* perform the worst in terms of recall and F1 score.

3.4.3.4 Experiments on Ingredient Prediction Given States

Many research has been conducted on ingredient retrieval and prediction in the past recent years. One state of the art model for ingredient prediction which is the baseline for our work is the inverse-cooking model for ingredient prediction proposed by [35]. We compare various versions



(a) Recall, precision, and F1 score for recipes with k number of states



(b) Individual recall, precision, and F1 score for each state

Figure 3.14. Recall, precision, and F1-score for recipes with k number of states and individual recall, precision, and F1 score for each state.

of our models with the this model and report the results in Table 3.7. As shown in Table 3.7, our models outperform the state-of-the-art ingredient-only approach proposed in [35].

We also have analyzed visual results from the Recipe1M dataset and compared ingredients predicted by our model, the baseline model, and the ground-truth as shown in Figure 3.15.

We also show results for state prediction by the embedding model in Figure 3.15. Although the states prediction is not perfect, the embedding used for the second step of the model is richer in context than the simple one-hot embedding. In other words, using the probability embedding rather than using the state one-hot tokens is helping the ingredient prediction model. Empirical experiments on feeding one-hot embedding one step at a time also agree with this inference.

Table 3.7. Results of the model for ingredients generation using state embeddings or word embeddings or both simultaneously.

Model	IOU	F1
Baseline [35]	31.57%	47.99%
States \rightarrow Ingredients	32.27%	48.79%
Tokens \rightarrow Ingredients	31.82%	48.28%
States+Tokens \rightarrow Ingredients	31.93%	48.4%
States \rightarrow Ingredients + Query	31.85%	48.31%
Ingredients + Query	32.19%	48.7%

3.5 Conclusion and Future Work

In this chapter, the state identification challenge is introduced for the first time, and a solution to it is provided using a deep convolutional approach. A state of an object is defined as the form an object could be transformed into, and the state identification challenge is defined as the problem of classifying an image of an object into its relative state. A useful dataset of cooking ingredients was gathered for the challenge. Using a proposed deep model based on Resnet [25], a promising level of accuracy was reached for state identification. We further tested our model on Imagenet images and semi-automatically provided state labels for images in Imagenet that are related to cooking ingredients. We showed that fine-tuning the model for each known object improves the average accuracy significantly. In future work, we will explore detection of all states inside an image, tracking the states in a video and providing continuous state labels for objects in a video.

The states of a cooking object are valuable information for a robot chef when performing cooking events and are closely related with the object itself. In Section 3.3 we presented a deep neural network with two heads and two joint losses for object and state classification. A language knowledge graph was deployed on top of confidences from a double loss CNN for extracting language based confidences. A MLP-based classifier was trained using the combination of confidences from both stages. Experiments on a state classification dataset consisting of cooking objects showed that using a language knowledge together with the confidences from the deep network improved both object and state classification performance. The results for this model show that objects and states are associated with each other and knowledge about either is important for identifying knowledge about the other.

	States	Ingredients
Ground-truth	<i>juice, powder, thick_liquid, whipped, minced, sliced, diced</i>	<i>flour, sugar, baking_soda, baking_powder, butter, milk, lemon_juice, eggs, cornstarch, lemon_zest, strawberries, brandy, cream</i>
Baseline [1]		<i>vinegar, salt, onion, olive_oil, <u>sugar, strawberries, crescent_rolls</u></i>
Our Model	<u>sliced, juice, powder</u> , <i>whole</i>	<i>salt, <u>sugar, butter, flour, baking_powder, eggs, milk,</u> vanilla_extract, <u>cream, strawberries</u></i>


	States	Ingredients
Ground-truth	<i>whipped, juice, jelly, baked</i>	<i>baking_soda, milk, instant_pudding, cake_mix, cool_whip, food_coloring</i>
Baseline [1]		<i>olive_oil, sugar</i>
Our Model	<i>creamy, <u>juice,</u> ground, <u>jelly</u></i>	<i>salt, <u>cool_whip, cake_mix, milk, wafer_cookie,</u> <u>instant_pudding, food_coloring</u></i>

Figure 3.15. Samples of states and ingredient prediction comparing our proposed model with the ground-truth labels. Depicted are cases that our model has performed much better than the baseline method. Underlined, green words are correct predictions from the model and the ground truth labels.

States of objects (ingredients) are valuable additional context in recognizing objects. Also correlation between various states of objects (e.g. juicy and chopped) for ingredients (e.g. water and onion) in a dish (e.g. soup) can be important in generating ingredients from a given image. In Section 3.4 we proposed a two-step model that incorporates a state-embedding to identify ingredients from a given image. State-embeddings were first created using the input cooking image through a deep transformer decoder which models the dependencies between the states. The second step, integrates the state embedding with the image embedding and uses ingredient and state embedding simultaneously to query the integrated features. Experiments prove that using a state embedding space is helpful in improving the ingredient prediction task.

States on ingredients are under-researched area of work. In our experiments we demonstrated how states of ingredients are correlated with each other, with other ingredients and motions and dish types. One path for future work is to identify states of ingredients in a video and track ingredient state changes through the video which is a very challenging and important task for robotic manipulation.

Chapter 4: Meal Image Understanding and Analysis

In previous chapters we explored how we can extract knowledge from cooking videos such as objects (e.g. tomato), motions (e.g. pouring), states (e.g. sliced) and functional units (events) and the entire task (making omelette). We used the Functional Object Oriented Network (FOON) knowledge representation jointly with deep convolutional and auto-regressive models to derive the knowledge and provide an understanding of a cooking video. Each image can be a rich source of information that can help augment the FOON knowledge representation and create task graphs for robotic manipulation purposes. In this chapter, we investigate knowledge extraction from a image representing a fully cooked meal. The ultimate goal of such a research would be to augment knowledge representation graphs such as FOON, task graph generation, or other side applications such as calorie estimation. Figure 4.1 depicts some of the knowledge that can be extracted from a meal image.

Previous research on cooking contents (i.e. meal images) has never addressed large scale analysis of all aspects of a cooking image (e.g. states, portions and calories) and there has been no large scale analysis of images to reproduce a task graph based on all elements of a given image. Also, in most research [91, 95], ingredients had to be visually recognizable in an image. In this chapter we want to explore large scale knowledge extraction of meal images on both visually and non-visually recognizable entities.

Cooking related applications have become a popular research area in recent years spanning from tasks such as ingredient recognition [35], dish classification [91, 95], recipe generation from a single image [99] to calorie estimation [136], and recipe retrieval [96]. Food nutrition, and health are two important aspects of our lives that require close monitoring and care and are strictly associated with cooking. Specifically, the amount of calorie intake in a meal is an important matter

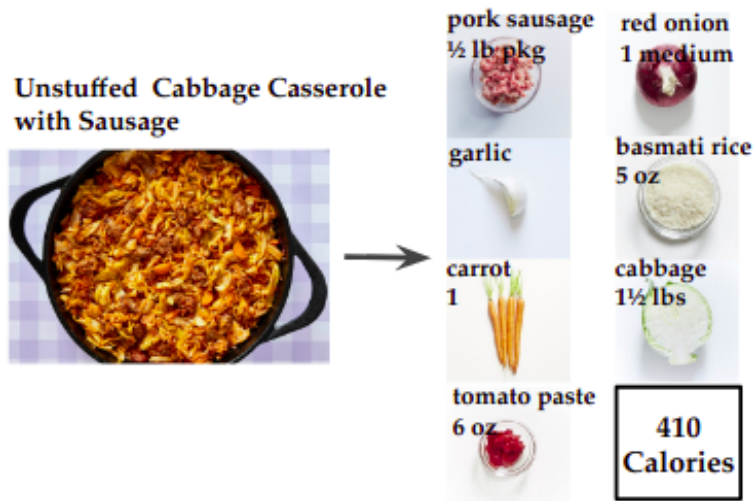


Figure 4.1. An example of meal kits generation from a single given prepared image of a meal. Meal kits generation includes generating ingredients, their portions and calorie.

of health. Many research have addressed calorie estimations from a single image, but they only use simple small sized datasets with a few ingredients or dish types [136, 137, 114]. They also lack simultaneous portion estimation of ingredients which can help improve calorie estimation, and reproduce the meal in different serving sizes.

In this chapter we propose a two stage pipeline. In the first stage, using a transformer based decoder [100], the main and optional ingredients of a meal (illustrated in the given image) are generated sequentially. In the second stage, all ingredients generated from the first stage are used for the task of total calorie estimation using a deep model with multiple encoder modules. Three encoders are deployed in the model to model per ingredient calories, units, and portions and the total calorie intake of the image. We finally introduce an application of this pipeline for meal kit generation (Figure 4.2).

4.1 Transformer Decoders

Transformers [100] are a type of deep models that were first proposed in 2017 by a team in Google for the task of machine translation. They were introduced as a replacement to the recurrent models used in machine translation to mainly improve parallelization at training time which also empirically resulted in better performance in machine translation.

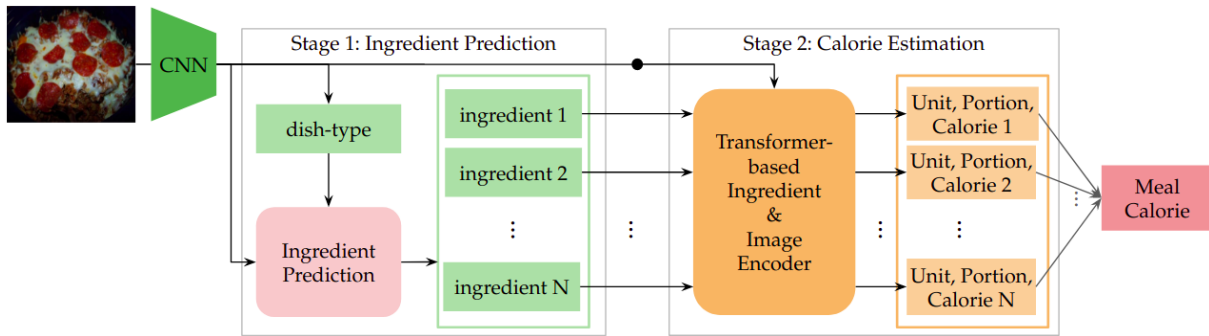


Figure 4.2. The two stage pipeline for calorie estimation. Stage 1: Ingredient set generation. Stage 2: Estimation of meal calorie using intermediate estimates of ingredient portions, units, and calories.

A transformer model at general level is similar to the recurrent based encoder-decoder architectures utilized in machine translation. It contains encoder stacks and decoder stacks that are connected in each layer. The connection flow is from an encoder layer to a decoder layer exactly similar to recurrent based encoder-decoder models for machine translation. Each encoder in the transformer contains n encoder layers and each decoder in the transformer contains n decoder layers (as shown in Figure 4.3.a). In our model we use the meal image as input and extract image embeddings. We use image embeddings and ingredient (output) shifted embeddings as input to our model as shown in Figure 4.3.b. As illustrated in 4.3.a, for the task of machine translation the input to the transformer decoder includes a positional encoding layer. The positional encoding captures order of input as it is required to be considered in such an application. In our application (i.e. ingredient set generation), the order of ingredient generation and in consequence the ingredient inputs is not important, therefore we remove the positional encoding from our proposed architecture. A more detailed description of our model is discussed in Section 4.3.2.

Each decoder layer in the transformer contains two stacked multi-head attention layers (as shown in Figure 4.4.b) and a layer normalization on top. The layers are connected through residual connections. A feed forward layer is applied as the last layer in a decoder layer to each position (ingredient embedding) in the network. Each multi-head attention also contains multiple scaled dot-product attention modules as shown in Figure 4.4.a. In Figure 4.4 The K stands for keys, Q stands for queries, and V stands for values. In our problem setting, ingredients are given

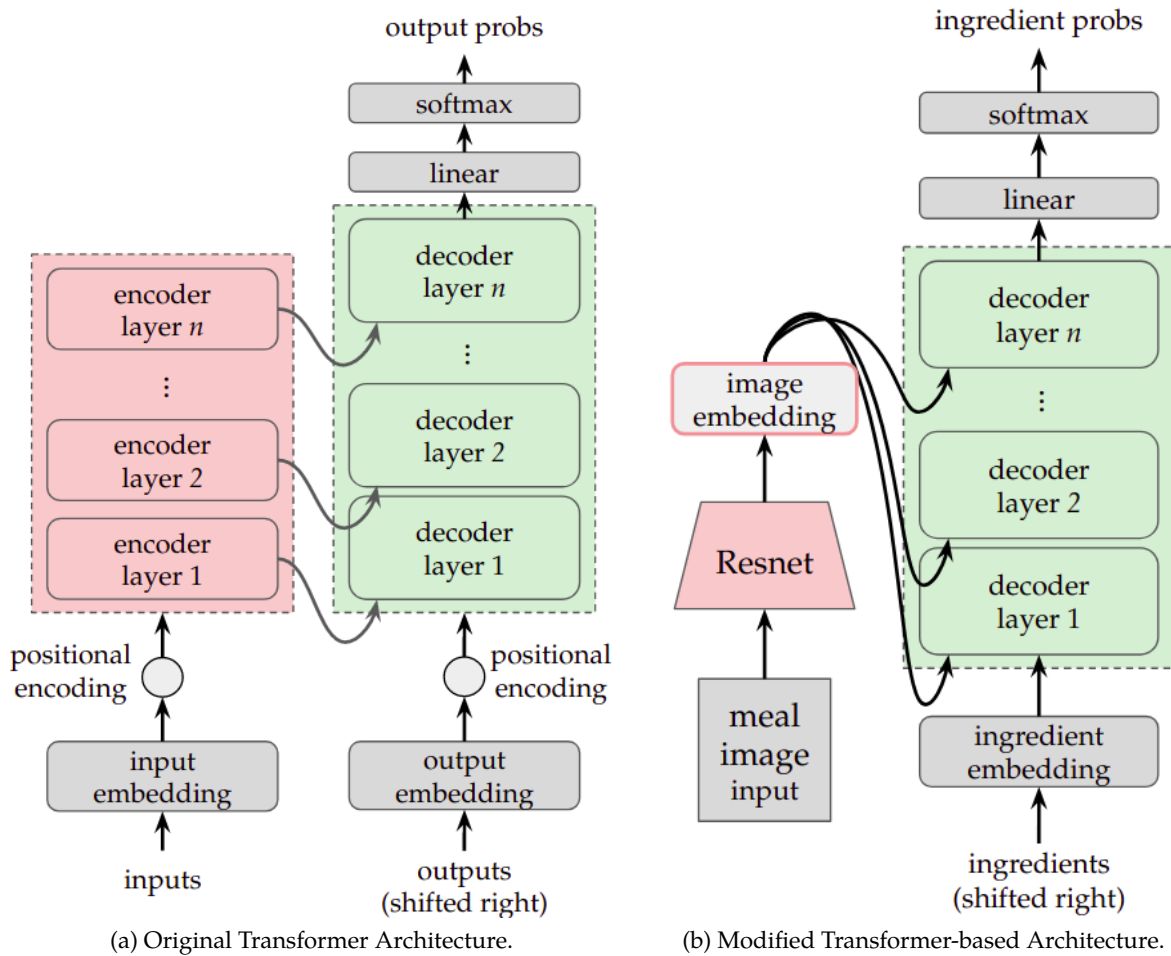
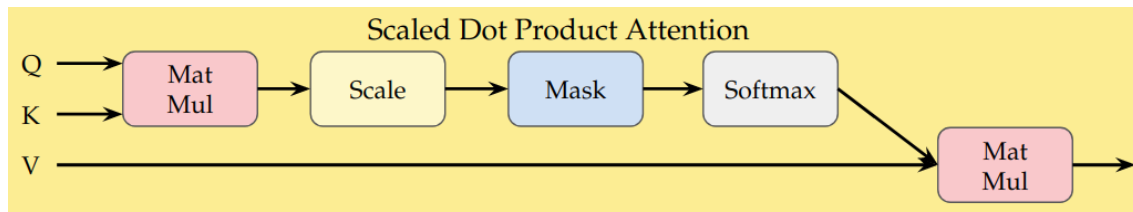


Figure 4.3. Illustration of the original transformer proposed for machine translation and the modified transformer based model for ingredient generation given a single image. We remove the positional encoding for ingredient set generation because we want to model order for ingredient generation.

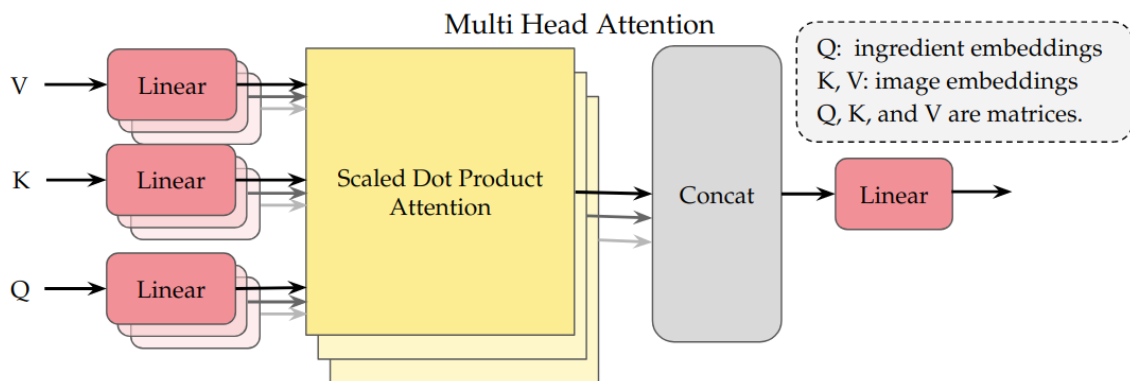
as queries to the model and image embeddings are given as keys and values. The self-attention can derive information associated with ingredients from images using this architecture.

4.2 Ingredient Generation

We propose a two-stage pipeline for calorie estimation of a given input image as shown in Figure 4.2. In the first stage (i.e. ingredient generation), given an image, the dish type, main ingredients and optional ingredients are predicted sequentially. In the second stage, using the image and the



(a) Scaled Dot Product Attention.



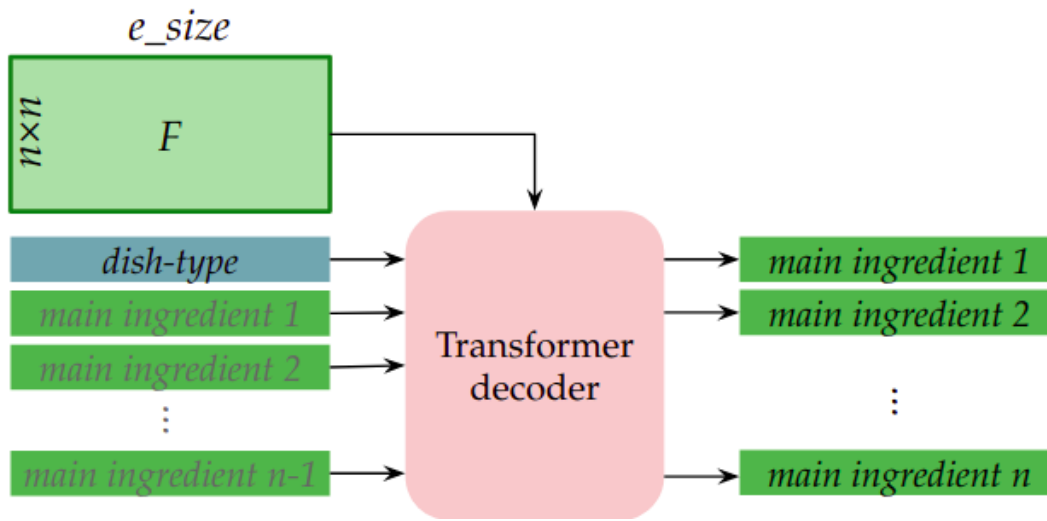
(b) Multi Head Attention.

Figure 4.4. Illustration of the details in the original Transformer such as scaled dot product attention and multi head attention.

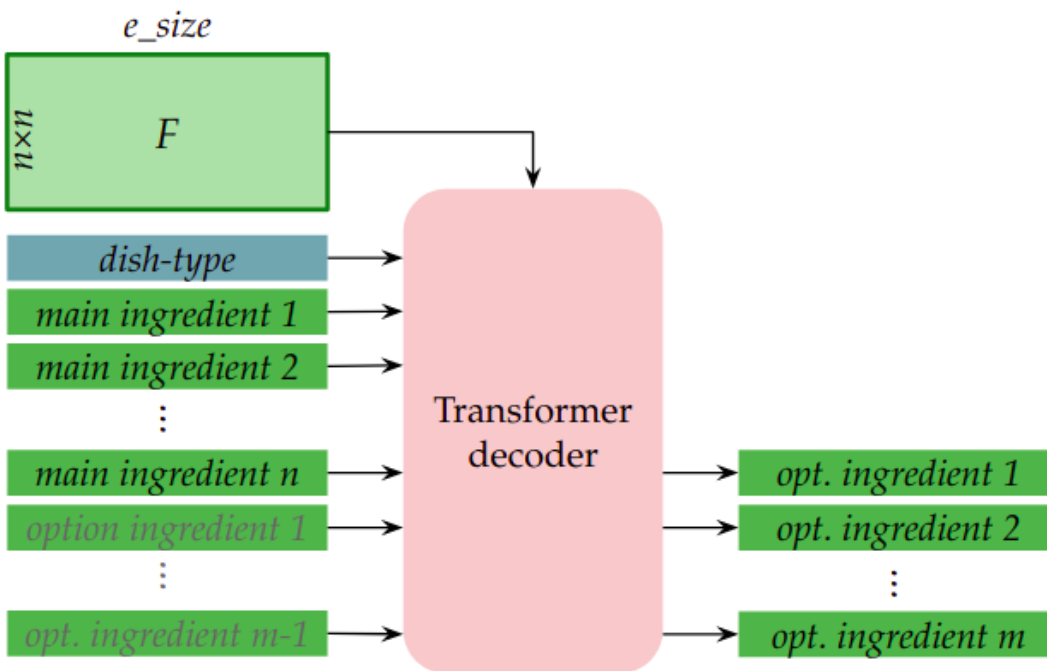
generated ingredients, an estimate of portions and calories of each of the ingredients is provided. Hierarchically the entire calorie intake of the given image is also estimated. In this section, we will discuss the first stage and in the next section the second stage (i.e. portion and calorie estimation) is elaborated.

4.2.1 Dish Classification

The first stage of the pipeline starts with an estimation of the dish type d , given the input image. A convolutional neural network (with a Resnet base) is trained to classify the image into one of N_d classes of dishes. We name the predicted dish as D . In the next steps, the predicted dish type is given along side the image as input to the ingredient generation model. The dish vocabulary is also combined with the ingredient vocabulary to provide token embeddings for dish names alongside ingredient embeddings.



(a) Predicting main ingredients iteratively (Main Ingredient Module).



(b) Predicting optional ingredients given main ingredients (optional Ingredient Module).

Figure 4.5. Prediction of ingredients given image features. A list of main ingredients are predicted in the first stage and afterwards a list of optional ingredients are predicted given the list of main ingredients. The list of main ingredients is predicted iteratively through iterations of correction.

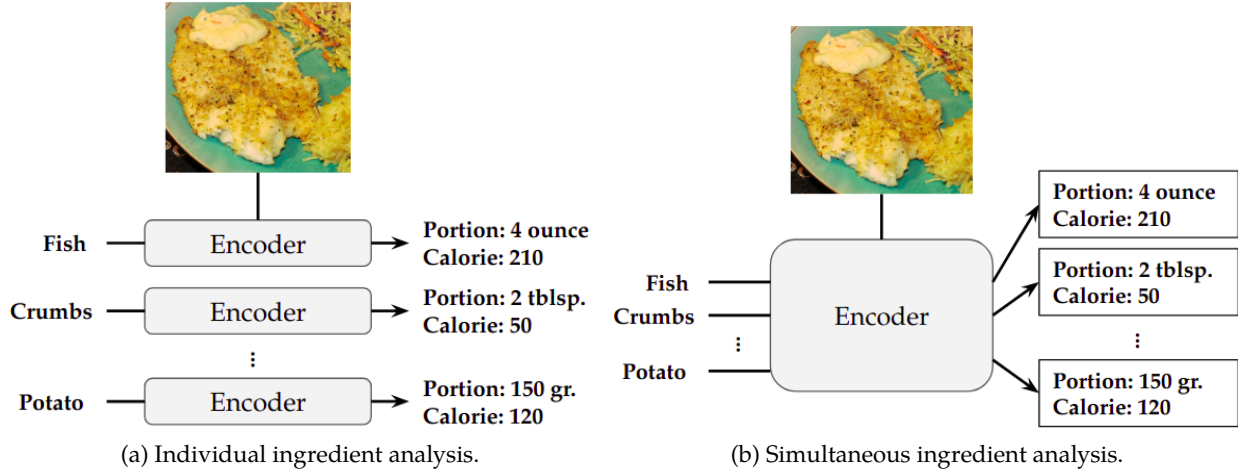


Figure 4.6. Difference between individual and simultaneous ingredient labeling given an image of a meal.

4.2.2 Ingredient Generation

The ingredient generation module utilizes a transformer decoder as in [100, 35] and performs ingredient generation in two steps. In the first step, the transformer generates a set of main ingredients, I^{main} , one step at a time as shown in the equation below and Figure 4.5.a. The model provides confidences for each ingredient at each step that can be useful for correcting wrongful selections.

$$I^{main} = p(I_{j+1}^{main} / I_j^{main}, F, D) \quad (4.1)$$

I_j^{main} and I_{j+1}^{main} are the j -th and $(j+1)$ -th generated ingredients, D is the predicted dish type from previous stage and F is the image features extracted from the last convolutional layer of a pre-trained Resnet mapped to $(n \times n) \times e_{size}$ where e is the embedding size. All ingredient and dish tokens are projected to embeddings of size e_{size} . The p is a transformer decoder that takes as input previous ingredients and image features and generates ingredients one step at a time as shown in Figure 4.5.a. In the second step, the transformer module generates a set of optional ingredients one step at a time given the main ingredients.

$$I^{opt} = p(I_{k+1}^{opt} / I_k^{opt}, I^{main}, F, D) \quad (4.2)$$

All generated ingredients are merged and used in next stages of the pipeline $I = I^{opt} + I^{main}$.

4.2.2.1 Formulation

The ingredient generation module takes image features and the predicted dish class, D , as input. Image features are extracted from the final convolutional layer, $V \in \mathbb{R}^{M \times n \times n}$. A 1×1 convolution layer and a reshape layer are applied to make the image features $F \in \mathbb{R}^{s_1 \times e_{size}}$.

$$F = \text{reshape}(\text{conv}_{1 \times 1}(V)) \quad (4.3)$$

The module also takes as input a one-hot token matrix comprising of a dish token and ingredients. Therefore, the vocabulary includes all ingredients in the dataset, N_i , combined with all dish vocabulary, N_d . Therefore, the dish classes would be considered in the input vocabulary but not in the output vocabulary making the total vocabulary size $N = N_d + N_i$. The input matrix of tokens is of size $I \in \mathbb{R}^{s_2 \times N}$ where N is the size of vocabulary and s_2 is the number of maximum ingredients the model accepts as input. The first token in I is always the predicted dish class D from the dish classification stage and the next tokens are generated ingredients from the previous step of the transformer. The token matrix is projected to an embedding matrix, $E \in \mathbb{R}^{s_2 \times e_{size}}$, through an embedding layer. The output of the transformer are generated ingredients at each step, $O_{ingr} \in \mathbb{R}^{s_2 \times N}$.

4.3 Ingredient Portion and Calorie Estimation

To extract richer knowledge we need to extract information about each of the ingredients identified from an image. Figure 4.6 shows two main ways we can extract information from an ingredient. Information can include portions (e.g. 4 ounce) of ingredients and the calorie ratio of an ingredient (e.g. 210). To estimate the portion we need to estimate a number (e.g. 4) and a label (e.g ounce) which are modeled as separate outputs but are connected through a connection at the end of the model. To extract information about an ingredient of a meal from an image there are two options:

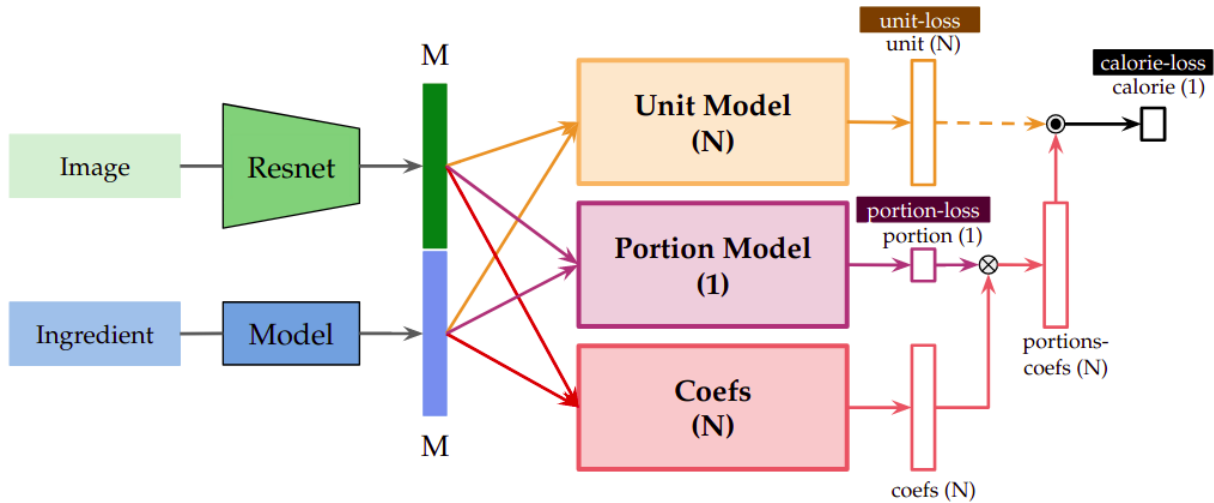


Figure 4.7. Model for individual ingredient analysis given a single image of a prepared meal. This model produces the calorie (or portion) of an ingredient associated with the given image.

- *Individual ingredient analysis*: Extract information of an ingredient independent of other ingredients as shown in Figure 4.6.a. A more detailed structure of this kind of a model is shown in Figure 4.7.
- *Simultaneous ingredient analysis*: Extract information of an ingredient in association with ingredients (Figure 4.6.b). A more detailed structure of this kind of a model is shown in Figure 4.8.

4.3.1 Individual Ingredient Analysis

To have a model that can take a single ingredient and image as input and create multiple outputs related to the ingredient we can re-formulate the problem where the ingredient would be a query and image would be the content base we are looking at. So, the model can be designed and implemented as an attention model as shown in Figure 4.7. The ingredients are first put into a dictionary of tokens and are assigned ids as in [138]. If the dictionary contains N vocabularies we would have ids between 0 to $N - 1$ assigned as shown below. The tokens ids are fed to an embedding layer to have a more compact feature representation of the ingredients.

$$vocabulary = \{potato : 0, onion : 1, tomato : 2, \dots, salt : N - 1\} \quad (4.4)$$

For the image input, we use the features from the last convolutional layer of a pre-trained Resnet architecture as shown in Figure 4.7. The image features go through a global average pooling layer and another fully connected layer to have a image feature vector of the same size as the ingredient feature vector. The cross modal feature vectors are concatenated and go through three different fully connected layers each for a separate goal:

- *Unit Model*: To extract the portion of an ingredient we need to know what the measurement metric is (e.g. is it ounce or pound). This fully-connected layer models units.
- *Portion Model*: This fully-connected layer models the portion for each ingredient.
- *Coefficient Model*: This fully-connected layer integrates portions with units to align them with each other and with calorie values for each ingredient in that meal.

The coefficient output tweaks the output from the portion layer and further the tweaked output is integrated with the unit output to create the calorie output for that ingredient as shown in Figure 4.7.

4.3.2 Simultaneous Ingredient Analysis

A two stream network is proposed to estimate the total calorie intake of a recipe (depicted in Figure 4.8) given the generated ingredients and image features. The first stream of the model is the calorie module that provides per ingredient calorie estimations. The second stream includes a unit module, a portion module and an alignment module. The unit module generates per ingredient unit predictions (e.g. teaspoon), the portion module uses the estimated units and the input ingredients to generate per ingredient portion estimations and the alignment module creates alignment between the generated units and portions using per ingredient calorie estimations. The model is trained end-to-end.

4.3.2.1 Inputs and Intermediate Outputs

The proposed model takes as input all generated ingredients I and image features V and contains two streams with four modules and three intermediate outputs. All encoders have two inputs; image features and ingredient embeddings. Image features are extracted from the last convolutional layer, $V \in \mathbb{R}^{M \times n \times n}$, and are projected and reshaped with a (1×1) conv layer to $F_c \in \mathbb{R}^{s_1 \times e_{size}}$, $F_u \in \mathbb{R}^{s_1 \times \frac{e_{size}}{2}}$, $F_p \in \mathbb{R}^{s_1 \times e_{size}}$, and $F_a \in \mathbb{R}^{s_1 \times e_{size}}$ for the calorie encoder, unit encoder, portion encoder, and alignment encoder respectively.

All encoders create intermediate embeddings. The calorie encoder, unit encoder, portion encoder, and alignment encoder create intermediate embeddings $E_c \in \mathbb{R}^{s_2 \times e_{size}}$, $E_u \in \mathbb{R}^{s_2 \times \frac{e_{size}}{2}}$, $E_p \in \mathbb{R}^{s_2 \times e_{size}}$, and $E_a \in \mathbb{R}^{s_2 \times e_{size}}$ respectively. The details of how these intermediate embeddings are generated are explained in Section 4.3.2.2.

Besides image features, each encoder has another set of inputs which is either ingredient embeddings or a combination of ingredient embeddings and intermediate embeddings from other encoders. The generated ingredients from stage one are converted to ingredient embeddings through an embedding layer and are fed to the calorie encoder, $I \in \mathbb{R}^{s_2 \times e_{size}}$. The original ingredient embeddings, I , are projected to smaller sized embeddings $I_u \in \mathbb{R}^{s_2 \times \frac{e_{size}}{2}}$ for the unit encoder. The input for the portion encoder, $I_p \in \mathbb{R}^{s_2 \times e_{size}}$, and alignment encoder, $I_a \in \mathbb{R}^{s_2 \times size}$, are created by concatenating the smaller sized ingredient embeddings, I_u , and the intermediate unit embeddings, E_u .

4.3.2.2 Encoder Structure

Each of the four modules include an *encoder* which follows the exact architecture of a transformer decoder [100] as explained in Section 4.1 and illustrated in Figure 4.3.b. The transformer decoder has multiple identical transformer decoder layers and takes as input three sets of matrices; queries, keys and values. Each of the transformer decoder layers in the encoder include two layers of multi-head attention layers with residual connections. The general formulation of each of the multi-head attention layers is shown in Equation 4.8.

$$\begin{aligned} \text{MultiHead}(Q, K, V) &= \text{Concat}(\text{head}_1, \dots, \text{head}_h)W^o \\ \text{where } : \text{head}_i &= \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \end{aligned} \quad (4.5)$$

where the projections are parameter matrices $W^o \in \mathbb{R}^{e_{size} \times s_2}$, $W_i^Q \in \mathbb{R}^{\frac{e_{size}}{h} \times s_2}$, $W_i^K \in \mathbb{R}^{\frac{e_{size}}{h} \times s_2}$, $W_i^V \in \mathbb{R}^{\frac{e_{size}}{h} \times s_1}$ where h is the number of heads, and *Attention* is the scaled dot product attention mechanism from [100] and shown in Equation 4.6.

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \quad (4.6)$$

where Q and $K \in \mathbb{R}^{\frac{e_{size}}{h} \times s_2}$, and $V \in \mathbb{R}^{\frac{e_{size}}{h} \times s_1}$ are projected matrices of queries, keys and values. In our model both queries and keys are a set of ingredient embeddings and values are image features. The last component of a transformer decoder layer is a position-wise feed forward network that is applied after the two multi-head attention layers [100]. A stack of transformer decoder layers makes a transformer decoder. The output of the transformer decoder is a matrix of embeddings $E \in \mathbb{R}^{s_2 \times e_{size}}$. Each of the intermediate embeddings (E_c, E_u, E_p, E_a) discussed in 4.3.2.1 are the output of a transformer decoder. For more details on transformer decoders readers are referred to [100].

4.3.2.3 The Final Network

The first stream of the network contains the calorie module and the second stream contains three modules (unit, portion and alignment encoders). The modules contain encoders that create intermediate embeddings. Calorie, portion, and alignment intermediate embeddings, E_c, E_p, E_a, E_u are projected to intermediate outputs through (per ingredient) identical fully connected layers, f_c, f_u, f_p, f_a .

$$\begin{aligned} o_c &= f_c(E_c) \\ O_u &= f_u(E_u) \\ o_p &= f_p(E_p) \\ o_a &= f_a(E_a) \end{aligned} \quad (4.7)$$

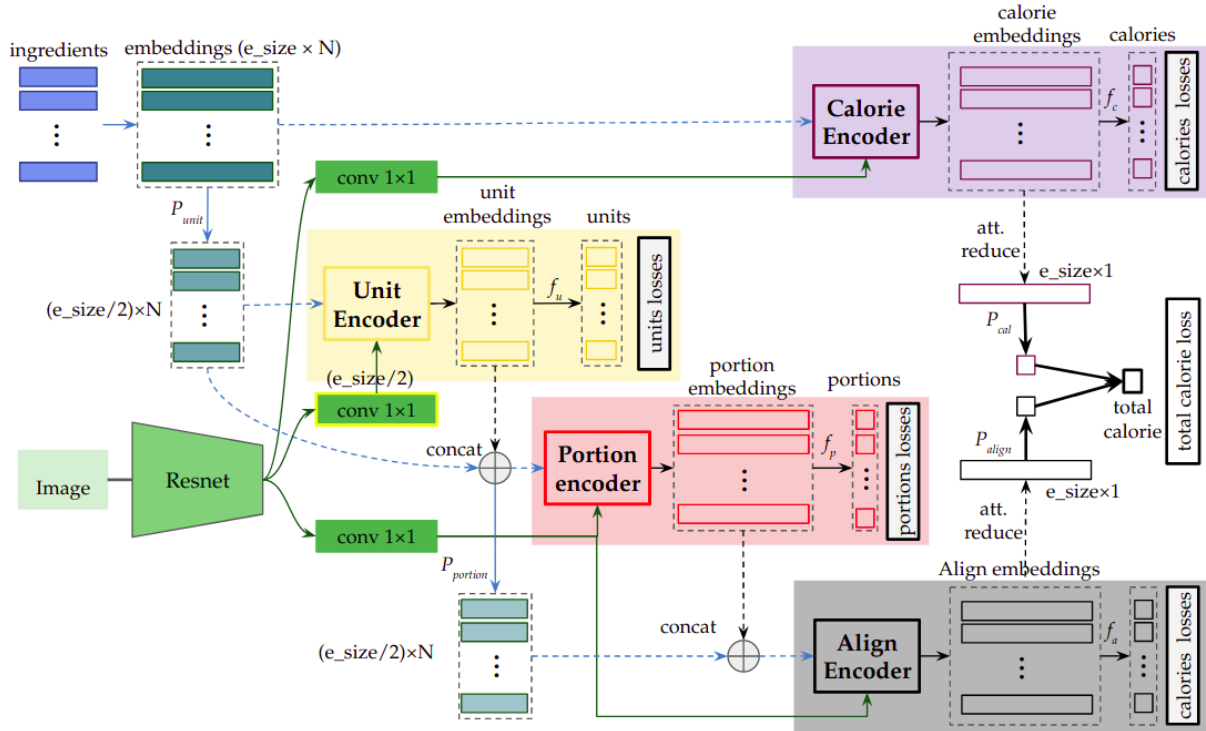


Figure 4.8. The calorie estimation model with individual ingredient units, portions, and calories incorporation.

where $o_c \in \mathbb{R}^{s_2 \times 1}$, $O_u \in \mathbb{R}^{s_2 \times N_{units}}$, $o_p \in \mathbb{R}^{s_2 \times 1}$, and $o_a \in \mathbb{R}^{s_2 \times 1}$ are intermediate calorie, unit, portion, and alignment outputs for s_2 ingredient inputs.

The combination of intermediate unit and portion outputs provides a representation of the amount of each ingredient. The purpose of adding an *alignment module* with the ingredient calories loss incorporated is to align unit and portions with the calorie values.

Attention reduction layers are applied to the calorie embeddings, E_c , from the first stream and alignment embeddings E_a from the second stream to create a reduced calorie vector $r_c \in \mathbb{R}^{1 \times e_{size}}$ and a reduced alignment vector $r_a \in \mathbb{R}^{1 \times e_{size}}$ respectively. Projections P_{cal} and P_{align} are applied to r_c and r_a respectively and their outputs are concatenated to produce the output of the model (total calorie estimate). This two stream (four modules) model is named T_{upc} henceforth.

4.3.2.4 Losses

Two types of loss are used in the entire model in five different locations. One MSE loss is used for ingredient calorie estimation in the first stream (L_1). The weighted cross-entropy loss is used for ingredient unit classification (L_2). Units with less frequency in the training data are assigned a larger weight for loss computation. Three MSE losses are used for portion estimation (L_3), calorie estimations in the alignment module (L_4) and total calorie estimation (L_5). The final loss is computed as below with $\lambda_1, \lambda_2, \lambda_3, \lambda_4, \lambda_5$ being hyperparameters.

$$L = \lambda_1 L_1 + \lambda_2 L_2 + \lambda_3 L_3 + \lambda_4 L_4 + \lambda_5 L_5 \quad (4.8)$$

4.4 Dataset Preparation for Analysis

In our experiments here we use the Recipe1M dataset created in [99]. Each instance in the dataset contains a dish name, a list of ingredient descriptions, and a list of recipes. An example instance in the Recipe1M dataset would appear as below with one or few meal images associated with it. The images and recipes were originally scraped from popular cooking websites and processed and parsed to extract relevant information from raw HTML [99].

- *Dish name*: Chocolate cake.
- *Ingredients*: 23 cup butter softened, 1 3/4 cups sugar, 2 large eggs, 1 teaspoon vanilla extract, 1 3/4 cups all-purpose flour, 3/4 cup cocoa powder, 1 1/2 teaspoons baking soda, 1 teaspoon salt, 1 1/2 cups yogurt, plain.
- *Recipe*:
 1. heat oven to 350f (180c).
 2. Grease and flour two 9-inch heart-shaped pans.
 3. In bowl, beat butter and sugar at medium speed of electric mixer until light and fluffy.
 4. Add eggs and vanilla ; beat well.
 5. Stir together flour , cocoa , baking soda and salt; add to butter.

6. Beat 3 minutes on medium speed.
7. Mix in yogurt.
8. Pour into prepared pans.
9. Bake 35 to 40 minutes or until wooden pick inserted in center comes out clean.
10. Cool 10 minutes ; remove from pans to wire racks.
11. Cool completely.
12. Frost and decorate as desired.

As shown in the example above the dataset originally does not have any explicit portion or state annotation. To prepare for our experiments, we derive states and portions from ingredient text through rule-based operations.

We train and evaluate our models in this chapter on the Recipe1M dataset [99], composed of 1,029,720 recipes scraped from cooking websites. The dataset contains 720,639 training, 155,036 validation and 154,045 test recipes, with a title, a list of ingredients, a list of cooking instructions and/or an image. In our experiments, we use only the recipes containing images, and remove recipes with less than 2 ingredients resulting in 252,547 training, 54,255 validation and 54,506 test samples [35].

Because the data was extracted by scraping cooking websites they are unstructured and include redundant ingredients. We follow all the operations in [35] (e.g. cluster 400 different cheese categories into one) and therefore reduce the number of ingredients from 16,823 to 1488. Some of the ingredients in [35] were clustered or split inaccurately. We performed a semi-automatic correction of some of the inaccurate clusters in [35]. For example we separated tomato from tomato sauce which were originally merged, or we merged sausages that were classified as separate classes with their brand names into one category. We only maintain high frequency ingredients (top 95%) which results in 202 ingredients. After this stage we maintain 132,442 train and validation recipes and 23,602 test recipes. We further automatically cluster recipes into 32 classes (i.e. dish names) using their ingredients list and recipe titles and remove outliers to keep 67,359 train and validation recipes and 11,743 test recipes. We extract ingredients (e.g. tomato paste) and their portions (e.g. 1 spoon) from the provided text for each ingredient in the dataset [99] and remove the recipes that

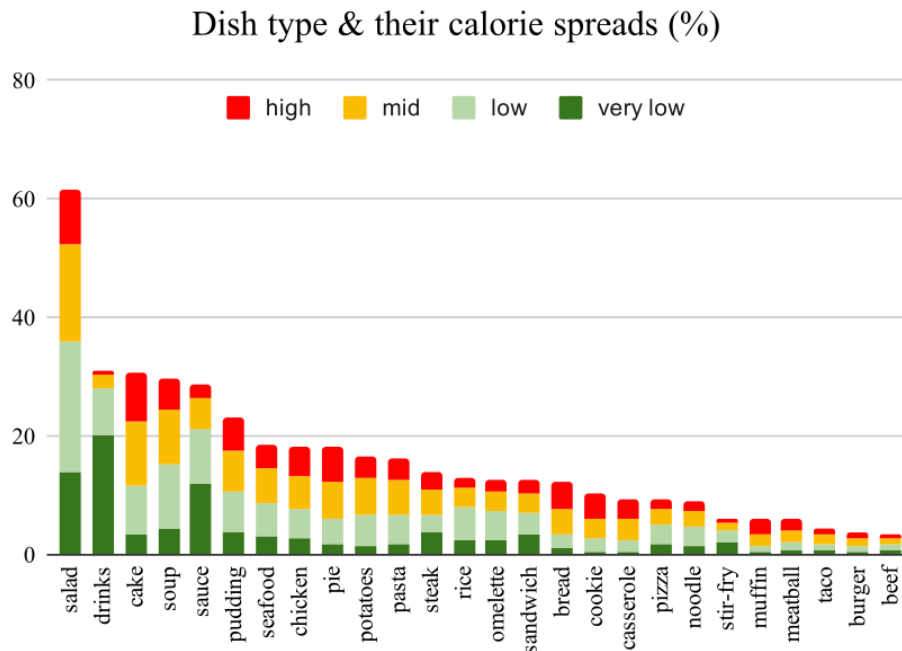


Figure 4.9. Frequencies of dish types in the Recipe1M dataset and their calorie spreads based on four different levels: high, mid, low, and very low.

contain extreme portion outliers or include ingredients with missing portions. The final dataset contains 42,455 train and validation recipes and 7,575 test recipes, with a title, a list of (ingredients, portions, units) tuples, and images. After all the operations we derive the frequency of each of the dish names and categorize all recipes into four different classes of calorie intake ranging from 0-500, 500-1000, 1000-1500, and 1500-2000 as shown in Figure 4.9 to observe the distribution of dish types and their calorie spreads.

4.4.1 Unit Identification and Sampling

We explored and parsed the Recipe1M dataset and identified any word that can be related to measuring ingredients. We identified 27 different measuring units as shown in Figure 4.11. We can see that many units (e.g. cup, tablespoon) are very frequent and many others (e.g. jar or bowl) are very rare. We keep the top 6 most frequent units in the dataset and convert all other frequent units to those units. For example *milliliter* or *liter* can easily be converted to *ounce*. Some other units such as *pinch* was also converted through its exact definition to *ounce*. We remove the very low

frequent units (e.g. box, bottle) without much loss in data. Looking at the top 6 frequent units we can observe how they can cover various sizes of ingredients. For large amounts of an ingredient (e.g. beef) we can use the unit *pound*, for mid-size ingredients (e.g. beans) we can use *ounce* and for smaller sizes of ingredients (e.g. salt) we can use *teaspoon* or *tablespoon*. Figure 4.10 shows the frequency of values for each of the top 6 units in the Recipe1M dataset.

4.5 Experiments and Analysis

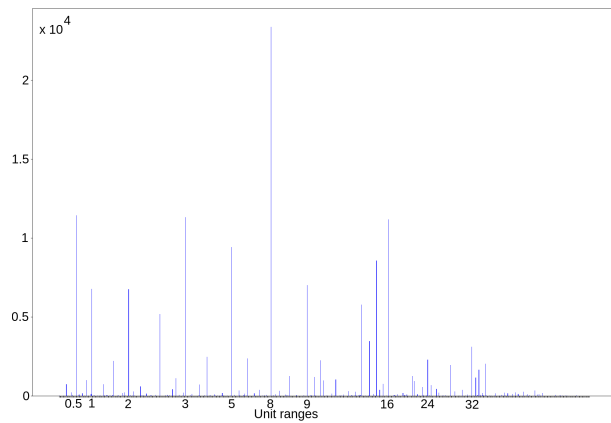
To analyze the model for meal image understanding we perform experiments on various models and evaluate the ingredients they generate as output and the auxiliary information (e.g. portions, units and calories) they predict for each ingredient.

4.5.1 Implementation Details

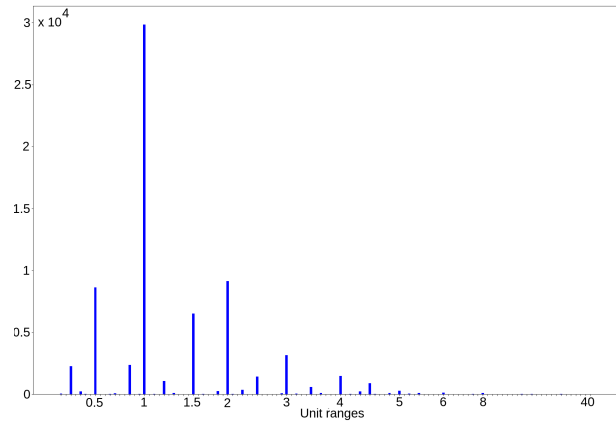
Images were resized to 256 pixels in their shortest side and random crops of 224×224 were taken for training. For evaluation the central 224×224 pixels were select. For the transformer encoders, we use a transformer with 2 blocks and 8 multi-head attentions, each one with dimensionality 64. For the last layer of the transformer, reduced embeddings of sizes 512, and 1024 were used. To obtain image embeddings we use the last convolutional layer of ResNet-50 model which would be of size $2048 \times 7 \times 7$ (i.e. $M = 2048, n = 7$ in Section 4.2). All the word embeddings and all transformer decoder input vectors were set to 1024. A maximum of 10 ingredients is used for each recipe. The models are trained with the Adam optimizer [139] for 60 epochs. Loss hyperparameters are all set to 1 with the exception of λ_4 being set to 0.1. All parts of the model are implemented with PyTorch. A GUI is implemented for user correction using Python and the Flask API.

4.5.2 Results

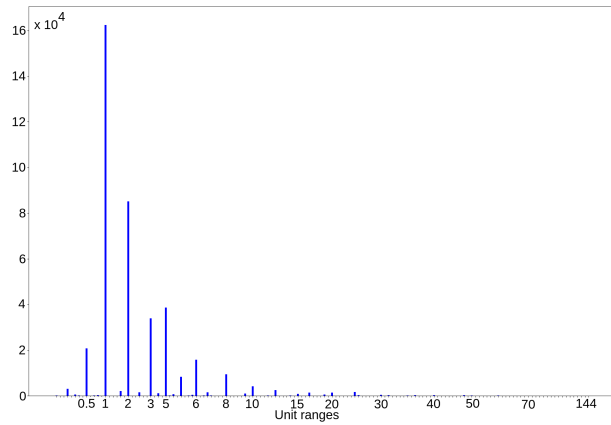
We perform ingredient generation in the first step of the pipeline. State of the art ingredient generation models [35] require much more improvements to be applicable to real world problems. Therefore, we proposed a multi-level model for ingredient generation. We also include semi-automatic user correction at each level to provide accurate ingredients for the second stage. For



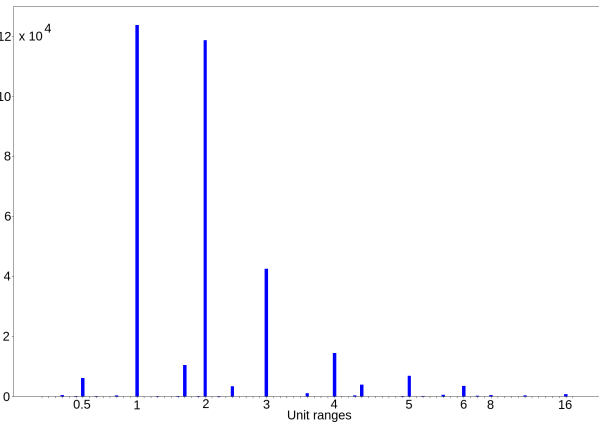
(a) Ranges of values for the ounce measurement



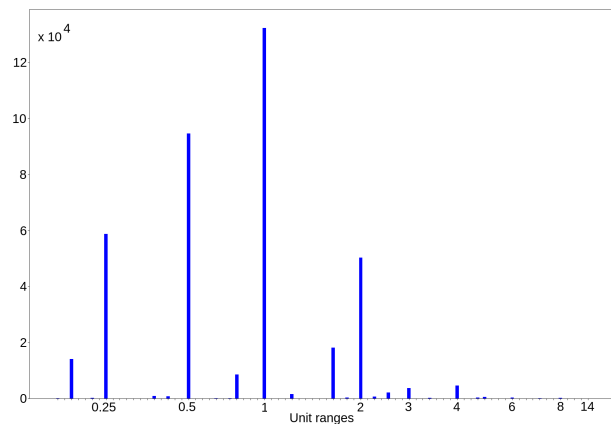
(b) Ranges of values for the pound measurement



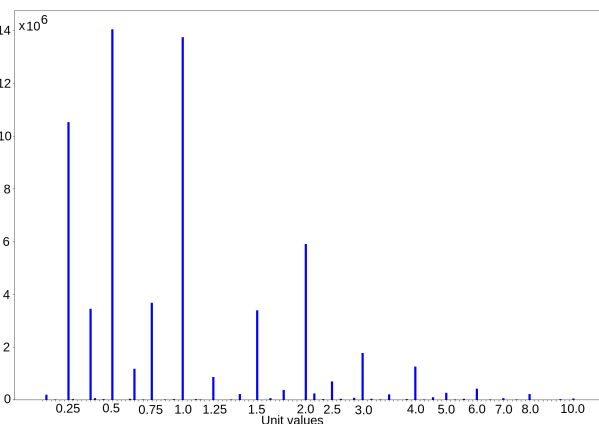
(c) Ranges of values for the counting measurement



(d) Ranges of values for the tablespoon measurement



(e) Ranges of values for the teaspoon measurement



(f) Ranges of values for the cup measurement

Figure 4.10. Ranges of values for most six frequent units in the Recipe1M dataset - ounce, pound, counting, tablespoon, teaspoon, and cup.

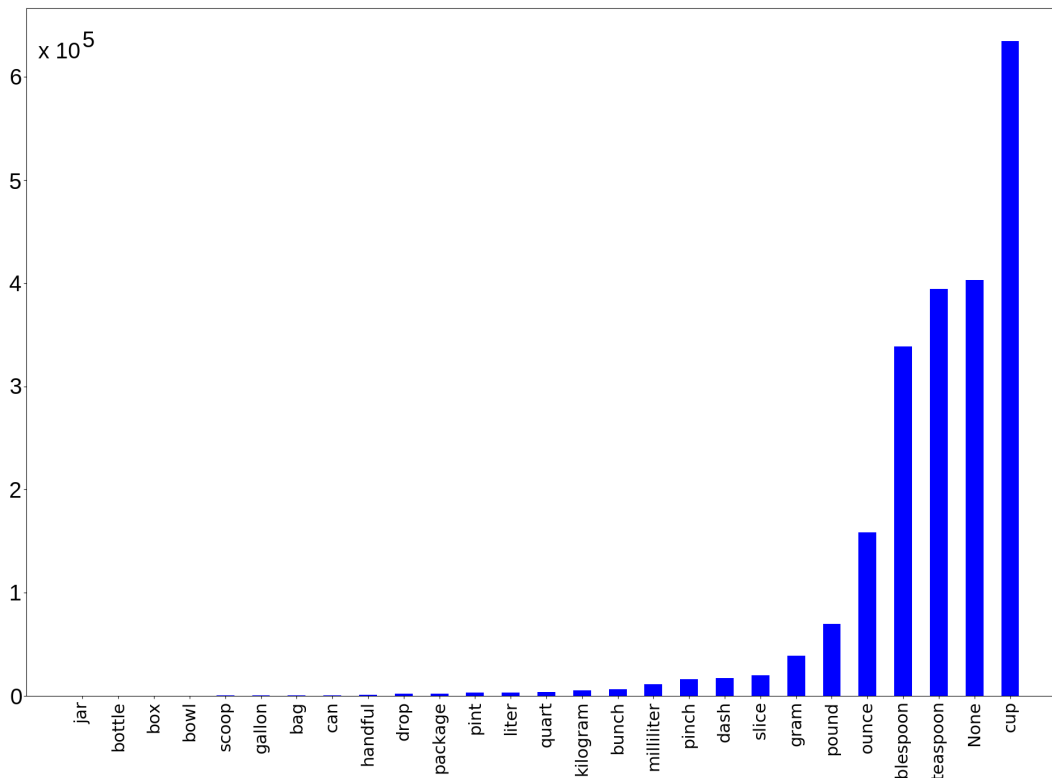


Figure 4.11. All measurements originally available in the Recipe1M dataset text (We used the top 6 frequent units and converted the other units to these six if possible).

this, we created an interface using Flask and obtain user feedback on our results at each level of the ingredient generation stage (<http://www.rpal-eve.cce.usf.edu/>).

4.5.2.1 Ingredient Generation

In the first step of the experiments we implemented a Resnet based CNN for dish classification and achieved near to 50% accuracy in classification of 32 dish types (e.g. omelette, cake, pizza, salad, etc). Ingredient generation is performed as suggested in Section 4.2. The ingredient vocabulary set comprises of 202 ingredients (e.g. butter, chicken, strawberry, flour, etc). The models for the ingredient generation include the dish types in their input vocabulary (i.e. 234 input tokens). Table 4.1 shows intersection over union (IoU) results for three different experiments with or without dish types given as input. Results for a) main ingredients generation: estimation of up to five main ingredients, b) optional ingredients generation: estimation of up to 10 optional ingredients given

Table 4.1. Intersection over union (IoU) for a) main ingredients, b) optional ingredients, and c) all ingredients generation.

	Given dish	No dish given
Main Ingredients	32.2%	27.3%
Optional Ingredients	49.2%	47.7%
All Ingredients	34.1%	31.5%

Table 4.2. Main ingredient generation results one step at a time (given N main ingredients. Accuracy is computed at each step (i.e. for each main ingredient)

Predicting	Dish name		Instances
	Given	Not given	
Predicted Main Ingredient No. 1	67.0%	56.8%	17393
Predicted Main Ingredient No. 2	55.3%	49.1%	15194
Predicted Main Ingredient No. 3	52.4%	44.4%	11460
Predicted Main Ingredient No. 4	45.3%	37.1%	7390
Predicted Main Ingredient No. 5	9.2%	21.9%	3803
Total	53.9%	47.1%	55240

an image and main ingredients and c) all ingredients generation is shown in the table. The results clearly show that when the model contains verbal knowledge about the dish type it estimates a more accurate ingredient list.

The ingredient estimates from the model for main ingredient generation is revised using user feedback. Table 4.2 shows results for estimating one ingredient at a time given the previous ingredients are revised by the user. As it can be observed in this table, the accuracy of generating main ingredients is much higher when revision happens. To evaluate revision accuracy, the revised ingredient is fed back to the model (and not the actual ground-truth ingredient at that time step). Therefore, the number of ingredients available at each time step in the test set is different for when the dish name is given in comparison to when it is not given as input. Furthermore, having the dish name as input for the model improves performance.

4.5.2.2 Portions and Units Estimation (Meal Kits)

The portion encoder creates estimates of portions of the listed ingredients by providing a value and a unit. The portions and units of an ingredient can be used to generate automatic meal kits procedures for a given image. This stage of the model is evaluated based on the mean absolute

Table 4.3. Mean absolute error of the portion estimation for different units compared with prior errors for each unit

Metric	Calorie MAE	
	Prior	Estimated
pound	323.7	162.8
ounce	238.7	155.1
cup	201.6	92.9
count	144.6	52.5
tblsp	99.4	68.9
tsp	7.6	5.3
total	144.2	78.5

error distance between the target and predicted portion and the accuracy of classification of units. In our experiments six different units as shown in Table 4.3 is used. In Table 4.3, MAEs for predicted portions for each individual unit and the prior MAE for each unit is shown. It can be observed that the calorie estimations are much better than the prior but the portions are not as good. The reasoning is that to have an accurate measurement of the amount of the ingredient an accurate combination of portion and unit is needed. The unit estimation accuracy of the model is 72.3%. In Figure 4.12, a few visual results of generated ingredients and their portions and calorie intakes for meal kits generation purposes is depicted.

4.5.2.3 Calorie Estimation

For calorie estimation, we evaluated the proposed model using mean absolute error (MAE) and percentage mean absolute error ($MAE\%$) on unseen test set. We compare the final model (T_{upc}) with a few variations of transformer based models. In one of the models we remove the ingredient based calorie training and only train the model for total recipe calories ($T_{calorie}$). In another version, we maintain the same base but we only individually generate calorie values for each ingredient and a final calorie of the entire recipe ($T_{calories}$).

We also compare the final model with a simple neural network for generating a calorie for each ingredient individually and adding them up to generate the final calorie ($NN_{calories}$) and a single ingredient based neural network based on portions, units and calories of each ingredient and adding them up to generate the final calorie (NN_{upc}).




	dish: Pizza		barbecue sauce	cheese	chicken	cilantro	onion	total		
	portion/unit	predicted	0.9 cup	5.5 oz	2.5 oz	1 tbsp	0.9 count	-		
		target	½ cup	8 oz	1 count	½ cup	½ cup	-		
	calorie	predicted	151	620	435	0	17	1385		
target		193	904	412	2	30	1540			
	dish: casserole		beef	onion	soup	milk	chilli	salt	total	
	portion/unit	predicted	0.9 lb	0.9 count	10.25 oz	1.25 cup	0.6 tsp	0.6 tsp	-	
		target	1 lb	1 count	10.75 oz	10.75 oz	0.5 tsp	0.5 tsp	-	
	calorie	predicted	1208	32	56	54	0	0	1431	
target		1136	44	79	128	1	0	1390		
	dish: Muffin		sugar	cinnamon	milk	egg	apple sauce	extract	blueberry	total
	portion/unit	predicted	0.8 cup	0.6 tsp	1 ½ cup	1.8 count	1 cup	⅓ tsp	1 cup	-
		target	2 tbsp	1 tsp	1 cup	2 count	¼ cup	1 tsp	1 ¼ cup	-
	calorie	predicted	329	0	80	154	96	0	94	629
target		114	12	103	156	41	14	106	548	

Figure 4.12. Examples of results from different parts of the end-to-end pipeline which includes predicted dish name, generated ingredients, portions and calorie estimates. The total predicted calorie intake and its ground-truth value is shown in the last column (gt: ground-truth).

We compare the model with a trained CNN on recipe calories (CNN), recipe calories estimated using prior calorie of each ingredient ($P_{i\text{mean}}$), and recipe calories estimated using prior dish names (P_{dish}).

We can observe from Table 4.4 that the transformer model which uses intermediate ingredient portion and calorie estimates performs the best in both predicting ingredient calorie estimation and recipe calorie estimation. Using ingredient based neural nets performs good but because the self attention between different ingredients is not modeled it performs worse than transformer based models. The CNN based model reaches to 49.8% $MAE\%$ for recipe calorie estimation showing the need for ingredient incorporation in this application. Just using the dish name (P_{dish}) and only knowing the ingredients ($P_{i\text{mean}}$) perform relatively worse than the proposed model in estimating meal and ingredient calorie.

Table 4.4. Results of calorie estimation based on mean absolute error and the percentage of error for the final transformer model and various other models

Model	Total Calorie MAE	
	MAE	MAE%
T_{upc}	279.4	37.5%
$T_{calorie}$	394.5	49.9%
$T_{calories}$	283.5	38.1%
NN_{upc}	306.7	39.7%
$NN_{calories}$	310	40.9%
CNN	380	49.8%
P_{imean}	323.3	44.7%
P_{dish}	407	52.3%

4.6 Discussion and Future Work

Most state-of-the-art models for analyzing cooking images and their relationships with ingredients focus on ingredient retrieval given an image [96, 97]. The models that generate ingredients (non-retrieval) from a given image have very low performance [35]. We incorporate a state-of-the-art method as base with a slightly modified (i.e. corrected) dataset and add semi-automatic correction into the model to enhance the ingredient generation model for meal kits generation. The semi-automatic model uses a web-interface which is discussed in Appendix 1. Our model also includes a stage (in the overall end-to-end pipeline) where portions and units (e.g. 1 spoon of oil) are generated for each ingredient using both attention between ingredients themselves and their underlying image features using transformer properties. An element in the portions generation stage that is unique to our model and beneficial is the containment of six unit types in the output which can automatically be mapped to meal kit content generation.

Another potential property of the model is the use of per ingredient unit and portion estimation to backtrack and provide re-computed ingredient amounts given a new serving amount of the meal. To our knowledge, this work is the first work on large scale calorie estimation on images with intermediate ingredient and portions estimation where the ingredients may or may not be (e.g. salt) visually seen in the image. Also, to prepare data for portions and calorie estimation we removed instances that lacked enough data from the Recipe1M dataset and therefore the dataset

used in the experiments is uniquely tailored for this application making it impossible to compare the results with any baseline methods.

The main objective of this chapter was extracting knowledge (e.g. ingredients, and portions) from a single image. Calorie estimation was also a side objective of the model. We proposed a pipeline using deep encoders to extract image features and generate ingredients (and their portions) from a given image. The ingredients are extracted using an auto-regressive encoder which captures both relative association between ingredients through self attention and ingredients and the image features through the transformer and Resnet features. The portions and furthermore calories of ingredients are also extracted with the assumption that knowing all ingredients contained in a recipe image can contribute to the portion knowledge base of a recipe (i.e. self-attention between portions of ingredients). The total calorie is estimated using all generated mid-level knowledge in an end-to-end manner. The current pipeline can semi-automatically generate ingredients with portions, and calories and a final calorie estimate of the entire recipe. In future work, we would like to extend experiments in a wider range of ingredients and recipes where more accurate portion estimates is available. We also would like to use this pipeline in robotic manipulation where using the predicted ingredients and their portions, state changes, and the task of making the recipe is inferred as a graph of sequential events.

Chapter 5: Concluding Remarks

To conclude, in this dissertation, we proposed various algorithms for knowledge extraction, inference and understanding of visual cooking contents. Visual cooking content in our research refers to images and videos of meals, or people preparing food. To extract knowledge from video, we need to use state of the art computer vision techniques namely deep learning algorithms. First, we proposed a pipeline that simultaneously incorporates deep convolutional networks and auto-regressive recurrent neural networks to achieve an understanding of a video representing a cooking task. We showed that it is important to model all aspects of a visual scene (e.g. objects and motions) explicitly while designing a deep model for video understanding. Therefore we used the Faster RCNN object detection algorithm to obtain an understanding of the objects and a recurrent neural network to model the motions of a video. We understood that although current state of the art deep architectures have improved significantly to their previous counterparts but they still have limitations in performance. Therefore we propose to utilize a knowledge representation graph called the Functional Object Oriented Network (FOON) [78, 2]. The knowledge representation bears object-object associations, object-motion inter-connections, and event-event order based relationships. We proposed a pipeline to integrate the FOON knowledge representation with state of the art deep models to improve video understanding performance. Through the pipeline we computed weights for objects and motions and identified the most relevant items to each step of the video. Results illustrated the significance of using the FOON knowledge representation for event and video understanding.

Although objects and motions are important for video understanding they are not complete sources of information. One other aspect of objects (i.e. ingredient) that is of significance for video understanding is the state of an object. Through a cooking process (e.g. preparing salad)

ingredients go through rounds of state change. A state (e.g. sliced) of an ingredient (i.e. tomato) can change the appearance of an object visually and therefore would effect the identification of an object. Besides, states of objects need to be known for a robot to make decisions for future manipulation tasks. Therefore we analyzed states of objects and introduced the state identification challenge. In conjunction with the challenge, we introduced a taxonomy of states for ingredients using knowledge acquired from FOON and we collected a labeled dataset of states for research. We discussed the importance of states analysis and proposed an initial model for state classification and joint ingredient and state classification on a single image. We further extended our experiments to identifying multiple ingredients and their states from a single cooking image and showed that how ingredients and states are intertwined and can effect the correct identification of each other in an image. Through our experiments we conclude that states are important for object (or ingredient) recognition especially in applications where the robot is performing manipulation tasks.

Finally, we explore the problem of knowledge representation extraction and augmentation from a single image. We discuss that to prepare a meal, we need to recognize everything about the meal, including the ingredients, states of ingredients and what portions of those ingredients are needed. We suggest that we can use a single cooking image from a fully prepared meal to identify information about the meal. Therefore, we introduce a model for total knowledge extraction (e.g. ingredients, states, and portions) from a single image. We propose a complex auto-regressive two-stage architecture for iterative ingredient recognition and per-ingredients state and portion estimation. We show by experiment that this model can be used for applications such as calorie estimation of a meal. Although this is the first time a model is presented for total knowledge extraction and calorie estimation in large scale, experiments show that more fine-grained labeled data of states, portions, and calories is required for improved performance.

5.1 Future Work

Our research and experiments identify problems and propose initial solutions to those problems. There are many paths of work that can be conducted for future experiments that can improve performance and get us closer to the ultimate goal of having an automatic cooking robot. One of

these paths is working on the knowledge representation. Enhancing the knowledge representation (i.e. FOON) by adding levels of entities or adding generalization measures [2] can improve video understanding. Additionally, in our experiments we only looked into states of ingredients in a static manner. For example, an ingredient is either sliced or un-sliced, or cooked or uncooked. Therefore, another route of work that can be explored is to dynamically track state change of an ingredient through the whole segment of a video. For example, in a video demonstrating a tomato being sliced, knowing how much of the tomato has been sliced at each point of the video may be important in tasks such as robotic manipulation. The problem of state tracking is even more relevant in the application of robotic manipulation, as the robot performing a task needs to know the state of the object at each time to make further decisions. We also suggest that knowledge extracted from a single image can be used for task tree generation. Therefore, a future work can be to use the model for image understanding in conjunction with the FOON knowledge representation to generate task trees of a meal. Finally, our work is a step towards identifying and introducing problems of visual content analysis in robotic manipulation that can be continued at a broader level in the future and progress towards building real service robots for home and outdoor applications.

References

- [1] A. B. Jelodar and Y. Sun. Joint object and state recognition using language knowledge. In *2019 IEEE International Conference on Image Processing (ICIP)*, pages 3352–3356, Sep. 2019.
- [2] D. Paulius, A. B. Jelodar, and Y. Sun. Functional object-oriented network: Construction & expansion. *International Conference on Robotics and Automation (ICRA)*, pages 1–7, May 2018.
- [3] A. B. Jelodar, D. Paulius, and Y. Sun. Long activity video understanding using functional object-oriented network. *IEEE Transactions on Multimedia*, pages 1–12, 2018.
- [4] Ronald Brachman and Hector Levesque. *Knowledge Representation and Reasoning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004.
- [5] D. Paulius and Y. Sun. A survey of knowledge representation in service robotics. *Robotics and Autonomous Systems*, 2019.
- [6] David Paulius and Yu Sun. A survey of knowledge representation and retrieval for learning in service robotics. *arXiv preprint arXiv:1807.02192*, 2018.
- [7] Christiane Fellbaum, editor. *WordNet: an electronic lexical database*. MIT Press, 1998.
- [8] K.K. Schuler. *Verbnet: A Broad-coverage, Comprehensive Verb Lexicon*. 2005.
- [9] Collin F. Baker, Charles J. Fillmore, and John B. Lowe. The berkeley framenet project. In *Proceedings of the 17th International Conference on Computational Linguistics - Volume 1, COLING '98*, pages 86–90, Stroudsburg, PA, USA, 1998.
- [10] Ahmad Babaeian Jelodar, Mehrdad Alizadeh, and Shahram Khadivi. Wordnet based features for predicting brain activity associated with meanings of nouns. In *Proceedings of the NAACL HLT 2010 First Workshop on Computational Neurolinguistics, CN '10*, pages 18–26, Stroudsburg, PA, USA, 2010.
- [11] Andrew Carlson, Justin Betteridge, Bryan Kisiel, Burr Settles, Estevam R. Hruschka, Jr., and Tom M. Mitchell. Toward an architecture for never-ending language learning. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI'10*, pages 1306–1313, 2010.
- [12] Matthew Richardson and Pedro Domingos. Markov logic networks. *Machine Learning*, 62(1):107–136, Feb 2006.
- [13] Qi Wu, Peng Wang, Chunhua Shen, Anton van den Hengel, and Anthony R. Dick. Ask me anything: Free-form visual question answering based on knowledge from external sources. *CoRR*, abs/1511.06973, 2015.

- [14] W. Min, B. K. Bao, S. Mei, Y. Zhu, Y. Rui, and S. Jiang. You are what you eat: Exploring rich recipe information for cross-region food analysis. *IEEE Transactions on Multimedia*, 20(4):950–964, April 2018.
- [15] Christopher Town. Ontological inference for image and video analysis. *Machine Vision and Applications*, 17(2):94, Mar 2006.
- [16] Nicolas Eric Maillot and Monique Thonnat. Ontology based complex object recognition. *Image Vision Comput.*, 26(1):102–113, January 2008.
- [17] Bernd Neumann and Ralf Möller. On scene interpretation with description logics. *Image Vision Comput.*, 26(1):82–101, January 2008.
- [18] X. Chen, A. Shrivastava, and A. Gupta. Neil: Extracting visual knowledge from web data. In *2013 IEEE International Conference on Computer Vision*, pages 1409–1416, Dec 2013.
- [19] Feng Niu, Ce Zhang, Christopher Ré, and Jude Shavlik. Elementary: Large-scale knowledge-base construction via machine learning and statistical inference. *Int. J. Semant. Web Inf. Syst.*, 8(3):42–73, July 2012.
- [20] Yuke Zhu, Alireza Fathi, and Li Fei-Fei. Reasoning about object affordances in a knowledge base representation. In *ECCV 2014*, pages 408–424, Cham, 2014.
- [21] Ranjay Krishna, Yuke Zhu, Oliver Groth, Justin Johnson, Kenji Hata, Joshua Kravitz, Stephanie Chen, Yannis Kalantidis, Li-Jia Li, David A. Shamma, Michael S. Bernstein, and Li Fei-Fei. Visual genome: Connecting language and vision using crowdsourced dense image annotations. *International Journal of Computer Vision*, 123(1):32–73, May 2017.
- [22] L. Herranz, S. Jiang, and R. Xu. Modeling restaurant context for food recognition. *IEEE Transactions on Multimedia*, 19(2):430–440, Feb 2017.
- [23] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.
- [24] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.
- [25] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778. IEEE Computer Society, 2016.
- [26] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1, NIPS’12*, pages 1097–1105, USA, 2012.
- [27] J. Wang, Y. Yang, J. Mao, Z. Huang, C. Huang, and W. Xu. Cnn-rnn: A unified framework for multi-label image classification. *CVPR*, 2016.

- [28] Jinseok Nam, Eneldo Loza Mencía, Hyunwoo J Kim, and Johannes Fürnkranz. Maximizing subset accuracy with recurrent neural networks in multi-label classification. In *NeurIPS*, pages 5413–5423. 2017.
- [29] Chih-Kuan Yeh, Wei-Chieh Wu, Wei-Jen Ko, and Yu-Chiang Frank Wang. Learning deep latent spaces for multi-label classification. In *AAAI*, AAAI 2017, pages 2838–2844, 2017.
- [30] T. Yao, Y. Pan, Y. Li, Z. Qiu, and T. Mei. Boosting image captioning with attributes. In *2017 IEEE International Conference on Computer Vision (ICCV)*, volume 00, pages 4904–4912, Oct. 2018.
- [31] Jeff Donahue, Lisa Anne Hendricks, Sergio Guadarrama, Marcus Rohrbach, Subhashini Venugopalan, Kate Saenko, and Trevor Darrell. Long-term recurrent convolutional networks for visual recognition and description. *CoRR*, abs/1411.4389, 2014.
- [32] J. Johnson, A. Karpathy, and L. Fei-Fei. Denscap: Fully convolutional localization networks for dense captioning. *CVPR*, 2016.
- [33] S. Venugopalan, L. A. Hendricks, M. Rohrbach, R. Mooney, T. Darrell, and K. Saenko. Captioning images with diverse objects. In *CVPR*, pages 1170–1178, July 2017.
- [34] Peter Anderson, Xiaodong He, Chris Buehler, Damien Teney, Mark Johnson, Stephen Gould, and Lei Zhang. Bottom-up and top-down attention for image captioning and visual question answering. In *CVPR*, 2018.
- [35] Amaia Salvador, Michal Drozdal, Xavier Giro-i Nieto, and Adriana Romero. Inverse cooking: Recipe generation from food images. In *CVPR*, June 2019.
- [36] Dhruv Mahajan, Ross Girshick, Vignesh Ramanathan, Kaiming He, Manohar Paluri, Yixuan Li, Ashwin Bharambe, and Laurens van der Maaten. Exploring the limits of weakly supervised pretraining, 2018.
- [37] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. Ssd: Single shot multibox detector. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *Computer Vision – ECCV 2016*, pages 21–37, Cham, 2016. Springer International Publishing.
- [38] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollar. Focal loss for dense object detection. In *ICCV*, pages 2999–3007, 2017.
- [39] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *NIPS*, pages 91–99, 2015.
- [40] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.
- [41] Pei-Yu Chi, Jen hao Chen, Hao-Hua Chu, and Jin-Ling Lo. Enabling calorie-aware cooking in a smart kitchen. In *PERSUASIVE*, volume 5033 of *Lecture Notes in Computer Science*, pages 116–127, 2008.

- [42] Xiatian Zhu, Chen Change Loy, and Shaogang Gong. Learning from multiple sources for video summarisation. *International Journal of Computer Vision*, 117(3):247–268, 2016.
- [43] Cornelia Fermuller, Fang Wang, Yezhou Yang, Konstantinos Zampogiannis, Yi Zhang, Francisco Barranco, and Michael Pfeiffer. Prediction of manipulation actions. *CoRR*, abs/1610.00759, 2016.
- [44] Patrick Perez and Ivan Laptev. Retrieving actions in movies. In *International Conference on Computer Vision (ICCV)*, pages 1–8. IEEE Computer Society, 2007.
- [45] Jeff Donahue, Lisa Anne Hendricks, Sergio Guadarrama, Marcus Rohrbach, Subhashini Venugopalan, Kate Saenko, and Trevor Darrell. Long-term recurrent convolutional networks for visual recognition and description. *CoRR*, abs/1411.4389, 2014.
- [46] Michele Merler, Bert Huang, Lexing Xie, Gang Hua, and Apostol Natsev. Semantic model vectors for complex video event recognition. *IEEE Trans. Multimedia*, 14(1):88–101, 2012.
- [47] Heng Wang and Cordelia Schmid. Action recognition with improved trajectories. In *International Conference on Computer Vision (ICCV)*, pages 3551–3558. IEEE Computer Society, 2013.
- [48] Guilhem Cheron, Ivan Laptev, and Cordelia Schmid. P-cnn: Pose-based cnn features for action recognition. *CoRR*, abs/1506.03607, 2015.
- [49] Bingbing Ni, Pierre Moulin, and Shuicheng Yan. Pose adaptive motion feature pooling for human action analysis. *International Journal of Computer Vision*, 111(2):229–248, 2015.
- [50] Cuiwei Liu, Xinxiao Wu, and Yunde Jia. A hierarchical video description for complex activity understanding. *International Journal of Computer Vision*, 118(2):240–255, 2016.
- [51] Hilde Kuehne, Juergen Gall, and Thomas Serre. An end-to-end generative framework for video segmentation and recognition. In *WACV*, pages 1–8. IEEE Computer Society, 2016.
- [52] Yuki Matsumura, Atsushi Hashimoto, Shinsuke Mori, Masayuki Mukunoki, and Michihiko Minoh. Clustering scenes in cooking video guided by object access. In *ICME Workshops*, pages 1–6. IEEE Computer Society, 2015.
- [53] Krishan Kumar, Deepti D. Shrimankar, and Navjot Singh. Eratosthenes sieve based key-frame extraction technique for event summarization in videos. *Multimedia Tools and Applications*, 77:7383–7404, 2017.
- [54] Krishan Kumar, Anurag Kumar, and Ayush Bahuguna. D-cad: Deep and crowded anomaly detection. In *ICCCT-2017*, 2017.
- [55] Ashesh Jain, Amir Roshan Zamir, Silvio Savarese, and Ashutosh Saxena. Structural-rnn: Deep learning on spatio-temporal graphs. *CoRR*, abs/1511.05298, 2015.
- [56] M. Hasan and A. K. Roy-Chowdhury. A continuous learning framework for activity recognition using deep hybrid feature models. *IEEE Transactions on Multimedia*, 17(11):1909–1922, Nov 2015.

- [57] K. Kumar and D. D. Shrimankar. F-des: Fast and deep event summarization. *IEEE Transactions on Multimedia*, 20(2):323–334, Feb 2018.
- [58] A. Gupta and L. S. Davis. Objects in action: An approach for combining action understanding and object perception. In *2007 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, June 2007.
- [59] A. Fathi and J. M. Rehg. Modeling actions through state changes. In *2013 IEEE Conference on Computer Vision and Pattern Recognition*, pages 2579–2586, June 2013.
- [60] Jianxin Wu, Adebola Osuntogun, Tanzeem Choudhury, Matthai Philipose, and James M. Rehg. A scalable approach to activity recognition based on object use. In *ICCV*, pages 1–8. IEEE Computer Society, 2007.
- [61] Jingjing Chen and Chong-Wah Ngo. Deep-based ingredient recognition for cooking recipe retrieval. In *ACM Multimedia*, pages 32–41. ACM, 2016.
- [62] Yang Zhou, Bingbing Ni, Richang Hong, Meng Wang, and Qi Tian. Interaction part mining: A mid-level approach for fine-grained action recognition. In *CVPR*, pages 3323–3331. IEEE Computer Society, 2015.
- [63] Xiaoyang Wang and Qiang Ji. Hierarchical context modeling for video event recognition. *IEEE Trans. Pattern Anal. Mach. Intell.*, 39(9):1770–1782, 2017.
- [64] Krishan Kumar and Deepti D. Shrimankar. Deep event learning boost-up approach: Delta. 03 2018.
- [65] K. Kumar, D. D. Shrimankar, and N. Singh. Event bagging: A novel event summarization approach in multiview surveillance videos. In *2017 International Conference on Innovations in Electronics, Signal Processing and Communication (IESC)*, pages 106–111, April 2017.
- [66] Xishan Zhang, Yang Yang, Yongdong Zhang, Huanbo Luan, Jintao Li, Hanwang Zhang, and Tat-Seng Chua. Enhancing video event recognition using automatically constructed semantic-visual knowledge base. 17:1–1, 09 2015.
- [67] C. F. Crispim-Junior, V. Buso, K. Avgerinakis, G. Meditskos, A. Briassouli, J. Benois-Pineau, I. Y. Kompatsiaris, and F. Bremond. Semantic event fusion of different visual modality concepts for activity recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(8):1598–1611, Aug 2016.
- [68] X. Chang, Z. Ma, Y. Yang, Z. Zeng, and A. G. Hauptmann. Bi-level semantic representation analysis for multimedia event detection. *IEEE Transactions on Cybernetics*, 47(5):1180–1197, May 2017.
- [69] M. Mazloom, X. Li, and C. G. M. Snoek. Tagbook: A semantic video representation without supervision for event detection. *IEEE Transactions on Multimedia*, 18(7):1378–1388, July 2016.
- [70] Fillipe D. M. de Souza, Sudeep Sarkar, Anuj Srivastava, and Jingyong Su. Temporally coherent interpretations for long videos using pattern theory. In *CVPR*, pages 1229–1237. IEEE Computer Society, 2015.

- [71] Fillipe D. M. de Souza, Sudeep Sarkar, Anuj Srivastava, and Jingyong Su. Spatially coherent interpretations of videos using pattern theory. *International Journal of Computer Vision*, 121(1):5–25, 2017.
- [72] Yu Sun and Shaogang Ren. Human-object-object-interaction affordance. In *WORV*, pages 1–6. IEEE Computer Society, 2013.
- [73] Yu Sun, Shaogang Ren, and Yun Lin. Object–object interaction affordance learning. *Robotics and Autonomous Systems*, 62(4):487–496, 2014.
- [74] Yu Sun and Yun Lin. Modeling paired objects and their interaction. In *New Development in Robot Vision*, pages 73–87. Springer, 2015.
- [75] A. Gupta, A. Kembhavi, and L. S. Davis. Observing human-object interactions: Using spatial and functional compatibility for recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(10):1775–1789, Oct 2009.
- [76] B. Yao and L. Fei-Fei. Modeling mutual context of object and human pose in human-object interaction activities. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 17–24, June 2010.
- [77] Dong Han, Liefeng Bo, and Cristian Sminchisescu. Selection and context for action recognition. *2009 IEEE 12th International Conference on Computer Vision*, pages 1933–1940, 2009.
- [78] David Paulius, Yongqiang Huang, Roger Milton, William D. Buchanan, Jeanine Sam, and Yu Sun. Functional object-oriented network for manipulation learning. In *IROS*, pages 2655–2662. IEEE, 2016.
- [79] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Computer Vision and Pattern Recognition (CVPR)*, 2015.
- [80] Christian Szegedy, Sergey Ioffe, and Vincent Vanhoucke. Inception-v4, inception-resnet and the impact of residual connections on learning. 02 2016.
- [81] B. Yao, X. Jiang, A. Khosla, A. L. Lin, L. Guibas, and L. Fei-Fei. Human action recognition by learning bases of action attributes and parts. In *2011 International Conference on Computer Vision*, pages 1331–1338, Nov 2011.
- [82] J. Wang, Y. Yang, J. Mao, Z. Huang, C. Huang, and W. Xu. Cnn-rnn: A unified framework for multi-label image classification. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2285–2294, June 2016.
- [83] C. Liu, Y. Cao, Y. Luo, G. Chen, V. Vokkarane, M. Yunsheng, S. Chen, and P. Hou. A new deep learning-based food recognition system for dietary assessment on an edge computing service infrastructure. *IEEE Transactions on Services Computing*, 11(2):249–261, March 2018.
- [84] Chang Liu, Yu Cao, Yan Luo, Guanling Chen, Vinod Vokkarane, and Yunsheng Ma. Deep-food: Deep learning-based food image recognition for computer-aided dietary assessment. In *Proceedings of the 14th International Conference on Inclusive Smart Cities and Digital Health - Volume 9677, ICOST 2016*, pages 37–48, New York, NY, USA, 2016.

- [85] L. Hou, Q. Wu, Q. Sun, H. Yang, and P. Li. Fruit recognition based on convolution neural network. In *2016 12th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD)*, pages 18–22, Aug 2016.
- [86] Jingjing Chen and Chong-Wah Ngo. Deep-based ingredient recognition for cooking recipe retrieval. In *ACM Multimedia*, pages 32–41. ACM, 2016.
- [87] Yoko Yamakata, Koh Kakusho, and Michihiko Minoh. Object recognition based on object’s identity for cooking recognition task. In *ISM*, pages 278–283. IEEE Computer Society, 2010.
- [88] Yuki Matsumura, Atsushi Hashimoto, Shinsuke Mori, Masayuki Mukunoki, and Michihiko Minoh. Clustering scenes in cooking video guided by object access. In *ICME Workshops*, pages 1–6. IEEE Computer Society, 2015.
- [89] Marcus Rohrbach, Anna Rohrbach, Michaela Regneri, Sikandar Amin, Mykhaylo Andriluka, Manfred Pinkal, and Bernt Schiele. Recognizing fine-grained and composite activities using hand-centric features and script data. *International Journal of Computer Vision*, pages 1–28, 2015.
- [90] Fillipe D. M. de Souza, Sudeep Sarkar, Anuj Srivastava, and Jingyong Su. Spatially coherent interpretations of videos using pattern theory. *International Journal of Computer Vision*, 121(1):5–25, 2017.
- [91] Yi Sen Ng, Wanqi Xue, Wei Wang, and Panpan Qi. Convolutional neural networks for food image recognition: An experimental study. In *International Workshop on Multimedia Assisted Dietary Management*, pages 33–41, 2019.
- [92] N. Martinel, G. L. Foresti, and C. Micheloni. Wide-slice residual networks for food recognition. In *WACV*, pages 567–576, March 2018.
- [93] Luis Herranz, Shuqiang Jiang, and Ruihan Xu. Modeling restaurant context for food recognition. *IEEE Transactions on Multimedia*, 19(2):430–440, February 2017.
- [94] R. Xu, L. Herranz, S. Jiang, S. Wang, X. Song, and R. Jain. Geolocalized modeling for dish recognition. *IEEE Transactions on Multimedia*, 17(8):1187–1199, Aug 2015.
- [95] Shota Horiguchi, Sosuke Amano, Makoto Ogawa, and Kiyoharu Aizawa. Personalized classifier for food image recognition. *IEEE Transactions on Multimedia*, 20:2836–2848, 2018.
- [96] Jing-jing Chen, Chong-Wah Ngo, and Tat-Seng Chua. Cross-modal recipe retrieval with rich food attributes. In *ACM International Conference on Multimedia*, MM 2017, pages 1771–1779, 2017.
- [97] 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 *The 41st International ACM SIGIR Conference on Research Development in Information Retrieval*, SIGIR 2018, pages 35–44, 2018.
- [98] Xin Wang, Devinder Kumar, Nicolas Thome, Matthieu Cord, and Frederic Precioso. Recipe recognition with large multimodal food dataset. In *ICMEW*, 2015.

- [99] Javier Marin, Aritro Biswas, Ferda Ofli, Nicholas Hynes, Amaia Salvador, Yusuf Aytar, Ingmar Weber, and Antonio Torralba. Recipe1m+: A dataset for learning cross-modal embeddings for cooking recipes and food images. *PAMI*, 2019.
- [100] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems 30*, pages 5998–6008. 2017.
- [101] A. B. Jelodar, M. S. Salekin, and Y. Sun. Identifying object states in cooking-related images. *arXiv preprint arXiv:1805.06956*, May 2018.
- [102] P. Isola, J. J. Lim, and E. H. Adelson. Discovering states and transformations in image collections. *CVPR*, 2015.
- [103] P. Pouladzadeh, S. Shirmohammadi, and R. Al-Maghrabi. Measuring calorie and nutrition from food image. *IEEE Transactions on Instrumentation and Measurement*, 63(8):1947–1956, 2014.
- [104] Austin Myers, Nick Johnston, Vivek Rathod, Anoop Korattikara, Alex Gorban, Nathan Silberman, Sergio Guadarrama, George Papandreou, Jonathan Huang, and Kevin Murphy. Im2calories: towards an automated mobile vision food diary. In *ICCV*, 2015.
- [105] P. Pouladzadeh, S. Shirmohammadi, and A. Yassine. Using graph cut segmentation for food calorie measurement. In *2014 IEEE International Symposium on Medical Measurements and Applications (MeMeA)*, pages 1–6, 2014.
- [106] Shaobo Fang, Fengqing Zhu, Chufan Jiang, Song Zhang, Carol J. Boushey, and Edward J. Delp. A comparison of food portion size estimation using geometric models and depth images. *2016 IEEE International Conference on Image Processing (ICIP)*, pages 26–30, 2016.
- [107] Hsin-Chen Chen, Wenyan Jia, Yaofeng Yue, Zhaoxin Li, Yung-Nien Sun, John D. Fernstrom, and Mingui Sun. Model-based measurement of food portion size for image-based dietary assessment using 3d/2d registration. *Measurement science technology*, 24 10, 2013.
- [108] T. Miyazaki, G. C. de Silva, and K. Aizawa. Image-based calorie content estimation for dietary assessment. In *2011 IEEE International Symposium on Multimedia*, pages 363–368, 2011.
- [109] Koichi Okamoto and Keiji Yanai. An automatic calorie estimation system of food images on a smartphone. In *Proceedings of the 2nd International Workshop on Multimedia Assisted Dietary Management, MADiMa 2016*, pages 63–70. Association for Computing Machinery, 2016.
- [110] Wataru Shimoda and Keiji Yanai. Cnn-based food image segmentation without pixel-wise annotation. In Vittorio Murino, Enrico Puppo, Diego Sona, Marco Cristani, and Carlo Sansone, editors, *New Trends in Image Analysis and Processing – ICIAP 2015 Workshops*, pages 449–457, 2015.
- [111] T. Ege, Y. Ando, R. Tanno, W. Shimoda, and K. Yanai. Image-based estimation of real food size for accurate food calorie estimation. In *2019 IEEE Conference on Multimedia Information Processing and Retrieval (MIPR)*, pages 274–279, 2019.

- [112] Parisa Pouladzadeh and Shervin Shirmohammadi. Mobile multi-food recognition using deep learning. *ACM Trans. Multimedia Comput. Commun. Appl.*, 13(3s), August 2017.
- [113] K. Aizawa, Y. Maruyama, H. Li, and C. Morikawa. Food balance estimation by using personal dietary tendencies in a multimedia food log. *IEEE Transactions on Multimedia*, 15(8):2176–2185, 2013.
- [114] P. Pouladzadeh, G. Villalobos, R. Almaghrabi, and S. Shirmohammadi. A novel svm based food recognition method for calorie measurement applications. In *2012 IEEE International Conference on Multimedia and Expo Workshops*, pages 495–498, 2012.
- [115] Takumi Ege and Keiji Yanai. Image-based food calorie estimation using knowledge on food categories, ingredients and cooking directions. In *Proceedings of the on Thematic Workshops of ACM Multimedia 2017, Thematic Workshops 2017*, pages 367–375, 2017.
- [116] M. Newman. *Networks: An Introduction*. OUP Oxford, 2010.
- [117] C. Adam Petri and W. Reisig. Petri net. *Scholarpedia*, 3(4):6477, 2008. revision 91646.
- [118] Gibson J J. *The theory of affordances, in Perceiving, Acting, and Knowing. Towards an Ecological Psychology*. Number eds Shaw R., Bransford J. Hoboken, NJ: John Wiley & Sons Inc., 1977.
- [119] T Brox, A Bruhn, N Papenberg, and J Weickert. High accuracy optical flow estimation based on a theory for warping. In *European Conf. on Computer Vision*, volume 3024, pages 25–36, Prague, Czech Republic, 2004.
- [120] Marcus Rohrbach, Sikandar Amin, Mykhaylo Andriluka, and Bernt Schiele. A database for fine grained activity detection of cooking activities. In *CVPR*, pages 1194–1201. IEEE Computer Society, 2012.
- [121] Tom Fawcett. An introduction to roc analysis. *Pattern Recognition Letters*, 27(8):861 – 874, 2006. ROC Analysis in Pattern Recognition.
- [122] A. B. Jelodar, M. S. Salekin, and Y. Sun. Identifying object states in cooking-related images. *arXiv preprint arXiv:1805.06956*, May 2018.
- [123] Yun Lin and Yu Sun. Task-oriented grasp planning based on disturbance distribution. In *Robotics Research*, pages 577–592. Springer, 2016.
- [124] Yun Lin and Yu Sun. Task-based grasp quality measures for grasp synthesis. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pages 485–490. IEEE, 2015.
- [125] Yun Lin and Yu Sun. Grasp planning to maximize task coverage. *The International Journal of Robotics Research*, 34(9):1195–1210, 2015.
- [126] Yun Lin and Yu Sun. Robot grasp planning based on demonstrated grasp strategies. *The International Journal of Robotics Research*, 34(1):26–42, 2015.

- [127] Yu Sun, Yun Lin, and Yongqiang Huang. Robotic grasping for instrument manipulations. In *Ubiquitous Robots and Ambient Intelligence (URAI), 2016 13th International Conference on*, pages 302–304. IEEE, 2016.
- [128] Yun Lin, Shaogang Ren, Matthew Clevenger, and Yu Sun. Learning grasping force from demonstration. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 1526–1531. IEEE, 2012.
- [129] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Fei-Fei Li. Imagenet: A large-scale hierarchical image database. In *CVPR*, pages 248–255. IEEE Computer Society, 2009.
- [130] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *ICLR*, May 2013.
- [131] R. Speer, J. Chin, and C. Havasi. Conceptnet 5.5: An open multilingual graph of general knowledge. *AAAI*, 2016.
- [132] C. D. Manning and H. Shutze. Foundations of statistical natural language processing. In *The MIT Press*, 1999.
- [133] Google. Google ngram viewer. <http://books.google.com/ngrams/datasets>, 2012.
- [134] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, undefinedukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NIPS*, 2017.
- [135] 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.
- [136] Pei-Yu Chi, Jen hao Chen, Hao-Hua Chu, and Jin-Ling Lo. Enabling calorie-aware cooking in a smart kitchen. In *PERSUASIVE*, volume 5033 of *Lecture Notes in Computer Science*, pages 116–127. Springer, 2008.
- [137] Wen Wu and Jie Yang. Fast food recognition from videos of eating for calorie estimation. In *2009 IEEE International Conference on Multimedia and Expo*, pages 1210–1213, 2009.
- [138] Yuke Zhu, Ce Zhang, Christopher Ré, and Li Fei-Fei. Building a large-scale multimodal knowledge base for visual question answering. *CoRR*, abs/1507.05670, 2015.
- [139] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

Appendix A: The Ingredient, Portions, and Calorie Estimation Web Interface

In chapter 4, we introduced a pipeline for knowledge extraction from a single image illustrating a cooked meal. The pipeline contains four steps that can be corrected in between by a user. We create a web interface at <http://www.rpal-eve.cce.usf.edu> that contains sequential web-pages for information generation and correction. The web-pages in this web interface are as follows:

- *Selecting the image* – The web-page which provides the user a button to browse and select a cooked meal for analysis.
- *Dish type selection* – The web-page providing a dish type from the output of the model and providing the chance for the user to correct it.
- *Main ingredient generation* – The web-page showing the main ingredients used in the given input image.
- *Optional ingredient generation* – The web-page showing the list of optional ingredients used in the image.
- *Seasonings generation* – The web-page showing the seasonings used in the given input image.
- *Per ingredient portion & calorie estimation* – The web-page that shows the estimated per ingredient portions and total image calorie.
- *Finalize* – Final webpage showing all the results.

A.1 Browsing and Selecting an Image

On the first page of the interface, the user is provided with a browse button. The user can use that button to upload an image to the server. This image is fed through the pipeline in the next

Image to Graph

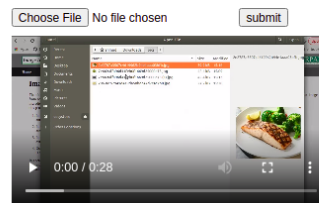
The ultimate goal for a robot cook is to visually look at an image and make that recipe from scratch. To do that the robot has to identify the ingredients and the amount of that ingredient used in the recipe. This project is defined to address this issues. Specifically, this project creates an incremental pipeline containing four stages to do this.

1. In the first stage given a single image the ingredients (e.g. tomato, onion) are automatically identified using a deep network.
2. In the second stage, given the image and ingredients, the states (e.g. sauce, sliced) are extracted using an attention based model.
3. In the third stage, units (e.g. gram) and portions (e.g. 200) of each ingredient is extracted using an attention based VQA like model.
4. Using these incrementally extracted information the recipe graph for each image is created which contains the steps to cooking that recipe.

In each of the incremental stages a semi-automatic refinement can be added to purify the results. This webpage provides this platform. You can start using this platform by uploading a recipe image on the right side of this page.

Select Image

To extract ingredients, states, portions and finally the recipe graph upload an image using the button below.



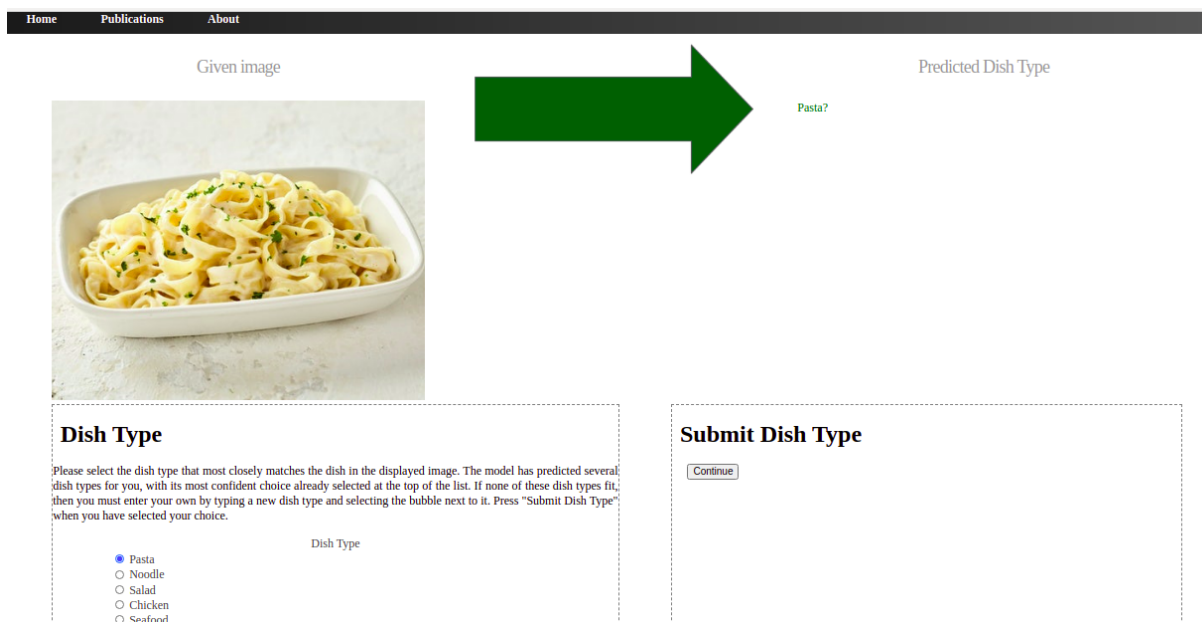
steps (web-pages). Also, on the first page, a demo video is presented for the user to understand how to use the web interface for ingredient, portion and calorie estimation. For information

A.2 Dish Type Selection

After submitting the image, the interface applies the convolutional network for dish classification and provides top $N(=5)$ dish types with the highest probabilities to the user in order in the dish type page. The dish type page provides radio buttons that allows the user to correct the dish type by choosing one of the other four dish types or writing through text another dish type.

A.3 Main Ingredient Generation

After estimating the dish type, and corrections from the user, the interface moves towards the main ingredient page. The first main ingredient is generated using the model proposed in Section 4.2, and shown to the user. The inputs for this model are the uploaded image and the corrected dish type from the previous page. The user can either correct this main ingredient or discard it and submit changes. The interface generates the second up to fifth main ingredient in the same manner allowing the user to correct the generated ingredients. After all main ingredients for the



uploaded image are generated, the model would generate an end token which is reflected as end of generation on the web interface. The user can further submit the results for the interface to move to the next page.

A.4 Optional Ingredient Generation


After the dish name and all main ingredients are generated and the user submits results, the model for optional ingredient generation is called. All optional ingredients are generated simultaneously given the image, dish type and main ingredients. The optional ingredients generated by the model include seasonings such as salt, and pepper. But, on the page for optional ingredients, seasonings are excluded. The user can remove or add optional ingredients of their choice in this page.

A.5 Seasonings Generation

To generate the seasonings used in an image no separate model is used. All the results generated by the model for optional ingredients is looked up in the seasonings table and identified. The

Home Publications About

Given image



Ingredient Portions

771 Calories

Main Ingredients

- Pasta - 1.0 pound
- Cheese - 2.5 ounce
- Milk - 0.25 cup

Ingredient Options

- Butter - 1.0 tablespoon
- Garlic - 2.0

Seasonings

- Salt - 0.25 teaspoon



seasonings page on the web interface, shows the seasonings to the user. Like previous pages, the user can perform corrections and submit the seasonings.


A.6 Per Ingredient Portion and Calorie Estimation

Before this page, dish type, all main ingredients, optional ingredients and seasonings are submitted. The model for portion and calorie estimation is now applied on the image and all generated ingredients and per ingredient portions and calories and the total calorie intake of the meal is estimated and shown to the user. The user is not to perform any corrections in this page but can change the unit of food intake. With this change the per ingredient portions and total calorie shown to the user changes on the web interface. The user will finalize the results and a final static version of all results is presented to the user.

Appendix B: Copyright Permissions

Permission from [3] for use of content in Chapters 2 is shown below.

Home Help Email Support Sign in Create Account



Long Activity Video Understanding Using Functional Object-Oriented Network
Author: Ahmad Babaeian Jelodar
Publication: IEEE Transactions on Multimedia
Publisher: IEEE
Date: July 2019
Copyright © 2019, IEEE

Thesis / Dissertation Reuse

The IEEE does not require individuals working on a thesis to obtain a formal reuse license, however, you may print out this statement to be used as a permission grant:

Requirements to be followed when using any portion (e.g., figure, graph, table, or textual material) of an IEEE copyrighted paper in a thesis:

- 1) In the case of textual material (e.g., using short quotes or referring to the work within these papers) users must give full credit to the original source (author, paper, publication) followed by the IEEE copyright line © 2011 IEEE.
- 2) In the case of illustrations or tabular material, we require that the copyright line © [Year of original publication] IEEE appear prominently with each reprinted figure and/or table.
- 3) If a substantial portion of the original paper is to be used, and if you are not the senior author, also obtain the senior author's approval.

Requirements to be followed when using an entire IEEE copyrighted paper in a thesis:

- 1) The following IEEE copyright/ credit notice should be placed prominently in the references: © [year of original publication] IEEE. Reprinted, with permission, from [author names, paper title, IEEE publication title, and month/year of publication]
- 2) Only the accepted version of an IEEE copyrighted paper can be used when posting the paper or your thesis online.
- 3) In placing the thesis on the author's university website, please display the following message in a prominent place on the website: In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of [university/educational entity's name goes here]'s products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to http://www.ieee.org/publications_standards/publications/rights/rights_link.html to learn how to obtain a License from RightsLink.

If applicable, University Microfilms and/or ProQuest Library, or the Archives of Canada may supply single copies of the dissertation.

[BACK](#) [CLOSE WINDOW](#)

Permission from [2] for use of content in Chapters 2 is shown below.



Functional Object-Oriented Network: Construction & Expansion

Conference Proceedings:
2018 IEEE International Conference on Robotics and Automation (ICRA)
Author: David Paulius
Publisher: IEEE
Date: May 2018

Copyright © 2018, IEEE

Thesis / Dissertation Reuse

The IEEE does not require individuals working on a thesis to obtain a formal reuse license, however, you may print out this statement to be used as a permission grant:

Requirements to be followed when using any portion (e.g., figure, graph, table, or textual material) of an IEEE copyrighted paper in a thesis:

- 1) In the case of textual material (e.g., using short quotes or referring to the work within these papers) users must give full credit to the original source (author, paper, publication) followed by the IEEE copyright line © 2011 IEEE.
- 2) In the case of illustrations or tabular material, we require that the copyright line © [Year of original publication] IEEE appear prominently with each reprinted figure and/or table.
- 3) If a substantial portion of the original paper is to be used, and if you are not the senior author, also obtain the senior author's approval.

Requirements to be followed when using an entire IEEE copyrighted paper in a thesis:

- 1) The following IEEE copyright/ credit notice should be placed prominently in the references: © [year of original publication] IEEE. Reprinted, with permission, from [author names, paper title, IEEE publication title, and month/year of publication]
- 2) Only the accepted version of an IEEE copyrighted paper can be used when posting the paper or your thesis online.
- 3) In placing the thesis on the author's university website, please display the following message in a prominent place on the website: In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of [university/educational entity's name goes here]'s products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to http://www.ieee.org/publications_standards/publications/rights/rights_link.html to learn how to obtain a License from RightsLink.

If applicable, University Microfilms and/or ProQuest Library, or the Archives of Canada may supply single copies of the dissertation.

BACK

CLOSE WINDOW

Permission from [1] for use of content in Chapter 3 is shown below.



Home

Help

Email Support

Sign in

Create Account



Joint Object and State Recognition Using Language Knowledge

Conference Proceedings: 2019 IEEE International Conference on Image Processing (ICIP)

Author: Ahmad Babaeian Jelodar

Publisher: IEEE

Date: Sept. 2019

Copyright © 2019, IEEE

Thesis / Dissertation Reuse

The IEEE does not require individuals working on a thesis to obtain a formal reuse license, however, you may print out this statement to be used as a permission grant:

Requirements to be followed when using any portion (e.g., figure, graph, table, or textual material) of an IEEE copyrighted paper in a thesis:

- 1) In the case of textual material (e.g., using short quotes or referring to the work within these papers) users must give full credit to the original source (author, paper, publication) followed by the IEEE copyright line © 2011 IEEE.
- 2) In the case of illustrations or tabular material, we require that the copyright line © [Year of original publication] IEEE appear prominently with each reprinted figure and/or table.
- 3) If a substantial portion of the original paper is to be used, and if you are not the senior author, also obtain the senior author's approval.

Requirements to be followed when using an entire IEEE copyrighted paper in a thesis:

- 1) The following IEEE copyright/ credit notice should be placed prominently in the references: © [year of original publication] IEEE. Reprinted, with permission, from [author names, paper title, IEEE publication title, and month/year of publication]
- 2) Only the accepted version of an IEEE copyrighted paper can be used when posting the paper or your thesis online.
- 3) In placing the thesis on the author's university website, please display the following message in a prominent place on the website: In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of [university/educational entity's name goes here]'s products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to http://www.ieee.org/publications_standards/publications/rights/rights_link.html to learn how to obtain a License from RightsLink.

If applicable, University Microfilms and/or ProQuest Library, or the Archives of Canada may supply single copies of the dissertation.

BACK

CLOSE WINDOW