August 2022

Pipeline for Calculating Calories for Print Recipes with Minimal User Intervention

Karl W. Holten
University of Wisconsin-Milwaukee

PIPELINE FOR CALCULATING CALORIES FOR PRINT RECIPES WITH MINIMAL USER INTERVENTION

by

Karl Holten

A Thesis Submitted in

Partial Fulfillment of the

Requirements for the Degree of

Master of Science

in Computer Science

at

The University of Wisconsin-Milwaukee

August 2022

ABSTRACT

PIPELINE FOR CALCULATING CALORIES FOR PRINT RECIPES WITH MINIMAL USER INTERVENTION

by

Karl Holten

The University of Wisconsin-Milwaukee, 2022
Under the Supervision of Professor Susan McRoy

Abstract: The thesis will provide a pipeline to estimate calorie counts from print recipes. The pipeline takes scanned recipes from cookbooks and uses Optical Character Recognition (OCR) to convert the scanned images of recipes to text. Several OCR tools were tested for their accuracy on fractions using a sample of the data, and the most accurate tool is used on the data. Next, a specially trained named entity recognition model is used to identify ingredients, quantities and units. These ingredients are used to search a database of values from the FDA to compute a calorie count for the recipe. The thesis tests the effectiveness of search by examining performance over 100 of the most common ingredients in the corpus of recipes. Finally, the thesis tests the performance of the model on a set of recipes, and found to estimate the calorie count at least as accurately as other automated approaches, such as those based on image recognition.

To my parents, Tom and Rose

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

BERT	Bidirectional Encoder Representations from Transformers
DSM	Dietary Self-Monitoring
FN	False Negative
FP	False Positive
GLUE	General Language Understanding Evaluation
IOB Tagging	Inside / Outside / Beginning Tagging
KNN	K Nearest Neighbors
NER	Named Entity Recognition
NLP	Natural Language Processing
OCR	Optical Character Recognition
ReLU	Rectified Linear Unit
TN	True Negative
TP	True Positive
USDA	United States Department of Agriculture

Thesis

1) Introduction

1.1) Problem and Purpose of Thesis

This thesis aims to help people improve their health by making it easier for them to track their consumption of calories when they prepare foods using recipes from printed books that do not provide a nutritional analysis. Obesity is a growing problem in the United States. Twenty years ago, no state had an obesity rate above 15%, but now only one state is below 20%, and two thirds of states have rates above 25% [1] Obesity increases the risk of various health conditions such as heart disease, stroke, diabetes, depression, and cancer. [2] There are a variety of techniques and strategies recommended by the National Institutes of Health to reduce obesity, one of which is to utilize a food log to track calorie consumption.[3]

Dietary self-monitoring (DSM) is considered a cornerstone of behavioral weight loss programs. DSM has been significantly linked with weight loss in a variety of studies, and is widely considered an effective strategy for weight loss. [5] Tracking frequency is also correlated with positive weight loss outcomes.[7] However, DSM as an approach has a number of flaws which must be noted. Tracking all foods is often considered burdensome and research has shown a significant increase in the number of incomplete records as more days of records are kept. Underreporting calorie consumption is also frequent. One study indicated obese users underreported their calorie consumption between 20-50%. [9]

Prior to the advent of computer technology, individuals used paper logs, but computer logs are now frequently preferred. A 2015 study by Hutchensson et al found that half the participants logging their calorie intake preferred computer recording, 44.4% preferred

smartphone and only 5.6% preferred paper-based records. [4] Burke et al have found that individuals utilizing applications for DSM rather than a paper log were significantly more adherent to logging. Over three months, adherents to applications had a 70% retention rate, versus a 30% retention rate for paper logging. [8] Hutchensson et al also found that the accuracy of calorie logging did not significantly differ between paper versus digital approaches. [4]

With the advent of computers and mobile phones, a variety of applications have been developed to assist with tracking calorie consumption. The functionality of these is discussed in detail in section 1.2. Some of these applications have tools to import online recipes and estimate their calorie intake. However, not all recipes are available online. In 2017, roughly 17.8 million cookbooks were sold in the US, showing that traditionally published recipes are still widely in use. [6] Many of these cookbooks are oriented toward health and dieting and could be a useful resource for weight loss. Currently, DSM for cookbook recipes using these applications requires the user manually enter the ingredients and their quantities. Developing an application to automate calculating calories and assist with tracking these recipes would possibly help address two issues: it may help improve the accuracy of self-reported calorie data, and it would help reduce the burden of using a calorie counting application.

The objective of this thesis is to produce a pipeline that takes images of cookbook recipes and convert them into accurate calorie counts with minimal user intervention. Accuracy will be determined by comparing the values from automatically processed images with those calculated by hand. Such an application could reduce the burden of DSM and help improve adherence to DSM and weight-loss. This pipeline starts by taking images of text from a selection

of cookbooks, and utilizing an OCR tool to convert images to text. Multiple OCR engines have been tested, and the most accurate has been selected for the application. Next, the pipeline uses a custom-trained named entity recognition model to label key information such as ingredients, quantities and units. The pipeline uses a vector space search on a database of ingredients to find calorie information, and calculates an estimated calorie amount for each ingredient. Finally, it sums the ingredient calories and calculates a final estimate per serving. This thesis will assess the rate of error in the application, comparing it a fixed benchmark of 85%. This benchmark is based on the rate of error for calorie counts estimated from state-of-the-art automated calorie estimation from images of food. [36]

1.2) Existing Applications and Their Accuracy

Many calorie tracking applications exist, with varying levels of recipe support. Many of these applications support adding recipes but require manually inputting the ingredients. Some examples of this type of applications include Lose It!, ControlMyWeight or LifeSum. Other applications have built in databases of recipes such as MyNetDiary, Applications such as MyFitnessPal support importing digital recipes from websites, but not doing OCR on print versions of recipes.

Recipe manager applications also have limited functionality to support automated ingestion of recipes from cookbooks. Many apps such as Cookmate, Master Cook, Mealime, Paprika 3 and Whisk do not support OCR. Some applications do support OCR in a limited fashion, such as Basil, Yummy, and The Cookbook App, but these applications require manual intervention from the user to draw multiple bounding boxes and do not support calorie

tracking. Some applications, such as BigOven, utilize data entry workers to type up recipes but this is not an automated solution.

Some applications, such as Yummly, utilize a camera to take pictures of the cooked recipe and attempt to estimate calorie values, however estimating calorie consumption based on photographs of food is a difficult problem. Estimating by picture also requires the user to create a recipe at least once before getting calorie information.

The accuracy of these applications varies based on their sophistication. However, many picture-based applications have accuracy ratings around 85%. Zhang's picture-based application had 85% accuracy over a controlled subset of 15 foods using standardized photography. [39] Okamoto and Yanai found "accuracy of image-based calorie content estimation shows that 79% of the estimates are correct within ±40% error and 35% correct within ±20% error" [38] GoCARB is another picture-based model which has an individual food accuracy rating of 85%. [40] Poply and Jothi also used a picture based approach, and found their accuracy on individual foods on items not used for training to be 85.74%. [41] The 85% mark will therefore be used as a guideline for the feasibility of the application.

An application that takes photos of recipes and converts them to estimated calorie values would be a novel and useful tool for tracking calories, as it could prove faster than a manual approach and more accurate than a photographic approach.

1.3) Architecture

The Print Recipe Calorie Calculator is a python application designed to prototype a workflow that could be utilized in a future mobile phone application. An outline of the pipeline

is displayed in Figure 1.1. We start with an image of a recipe. In this prototype, the images are provided from a corpus of recipes from the Cookbook and Home Economics Collection from archive.org. [28] The image is processed via Amazon Textract OCR to obtain a text version. After this, key elements required to calculate the calories for a recipe (such as ingredients, units, and quantities) are extracted from the text using a specially trained named entity recognition model. Next, quantities, units and ingredients are grouped together, accounting for special cases such as countable items and unquantified amounts. Food information from the USDA FoodData Central database has been adapted into a data file for search. Each query searches the database using cosine similarity and then applies a weight based on the distribution of ingredients across the corpus. Calories per unit are pulled from the database and combined to create an estimate of the calorie value of the recipe, which is displayed for the user.

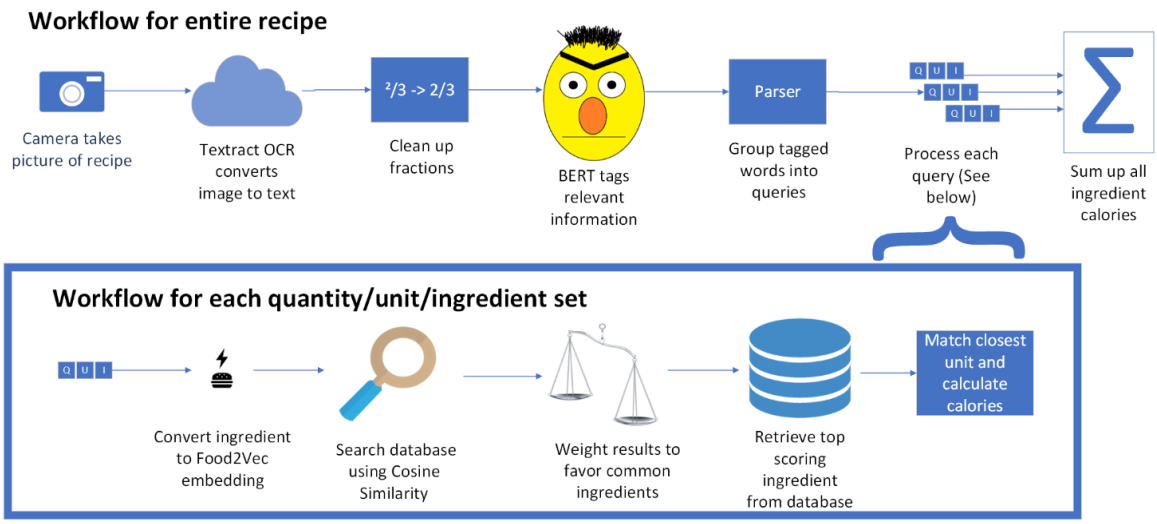

Figure 1.1 Overview of application pipeline

2) Background

2.1) OCR Applications and Their Architecture

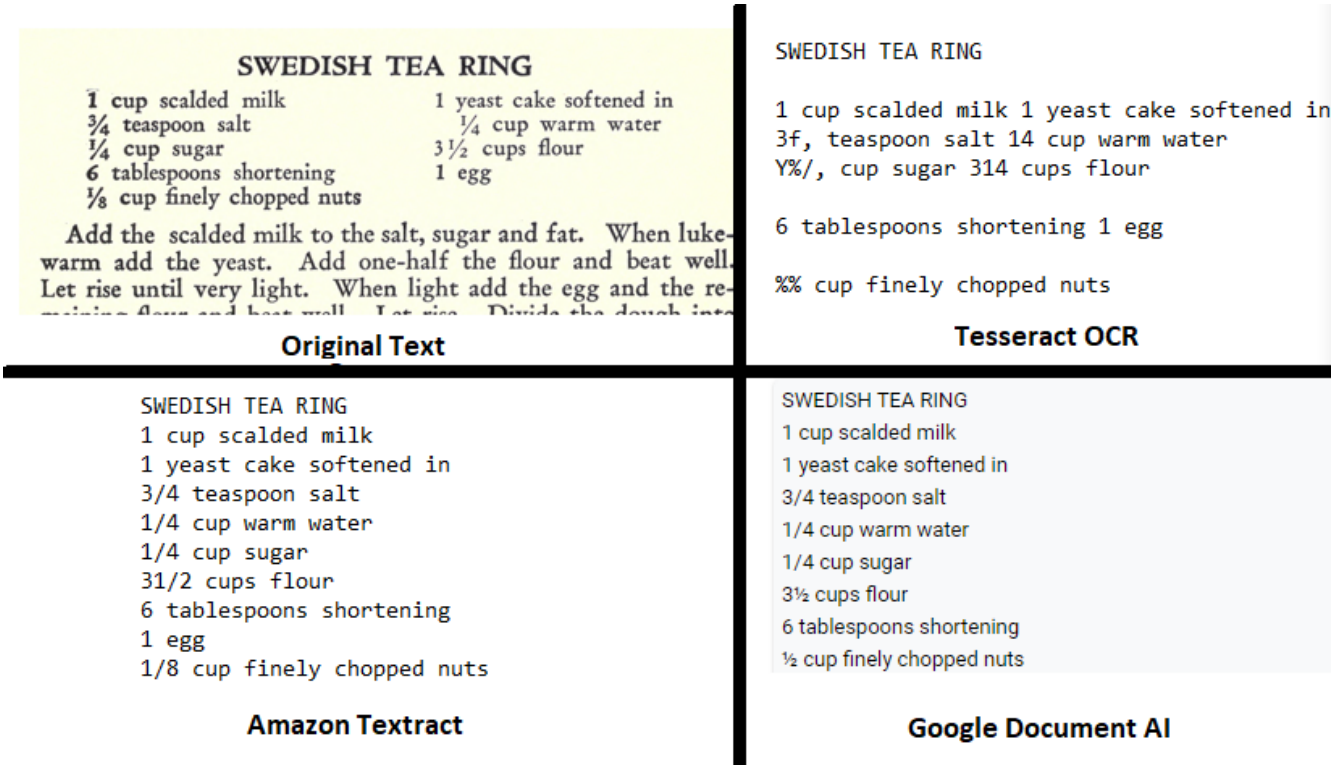

Figure 2.1 Comparison of one recipe over multiple OCR solutions.

Several off the shelf solutions were considered for the OCR component of the application. Side-by-side comparisons between the performances can be seen in figure 2.1. Two of these solutions are proprietary and one is open source. Less information is publicly available about the inner workings of the proprietary solutions, so they will be discussed first.

Both Amazon Textract and Google's Document AI advertise the ability to do both OCR and entity linking. Amazon Textract was released in May 29th, 2019. Amazon's vice president of machine learning, Swami Sivasubramanian, stated: "The power of Amazon Textract is that it accurately extracts text and structured data from virtually any document with no machine

learning experience required. Subsequently, developers can analyze and query the extracted text and data using our database and analytics services." [12] A similar proprietary product is Google's Document AI, released in November 2020. Document AI also promises to perform OCR and entity linking, by "transforming documents into structured data" using automation. [13] However, both applications are aimed as business audiences, and their entity linking functionality was found to be targeted toward table-based information. While these solutions are useful for OCR, their entity linking components are not useful for calorie calculation.

The open-source program Tesseract tells us more about the workings of OCR. Tesseract is an OCR solution originally developed by Hewett-Packard between 1985 and 1995. It was revised by Google and released into the public in 2005 and since then has been maintained by an open-source community. [10] Tesseract was not selected as the OCR solution for this application, but given the proprietary nature of the other two OCR solutions, understanding its architecture gives us an insight into how a modern OCR program works. While Google and Amazon do not publicize the details of their algorithms, articles from Google research staff have released articles discussing the various lessons learned from Tesseract, which have presumably influenced their own OCR pipeline. [17] Tesseract's architecture was explained in a research paper on release, and follows a step-by-step pipeline.

The first step Tesseract performs is binarization. Binarization is a form of preprocessing which turns color and greyscale images into black and white ones. This is done as a way of simplifying the task of matching characters, since matching pixels can be turned into a binary judgement of yes/no rather than trying to compare shades of grey. The simplest method for doing binarization is to set a threshold where greyscale pixels below a certain saturation value

are considered white and images above the threshold are considered black. There are a variety of ways to calculate this threshold. For example, the Otsu binarization method finds the median in a histogram of all greyscale saturation values in the image. Local binarization algorithms consider an N-by-N window of pixels and calculate a saturation threshold based on that window instead of the entire document. As a general rule, global methods perform better on greyscale images and local ones on colored images or ones under intense illumination. [16] Tesseract uses a local binarization technique.

Next, Tesseract performs a page layout analysis. First, Tesseract detects vertical lines and images in the binarized document. Tesseract uses the concept of tab indented areas to help determine possible groupings for text. [18] Tesseract groups together tab-stop areas of text as blobs. The blobs are then refined by performing connected components analysis, which is a task designed to assign groups to clusters of pixels. Once this is done, the blobs are ready to be organized into text lines and words. [18]

Multiple passes are made over lines to determine if the text is in a fixed pitch font or if it is a proportional font in a process called character segmentation. Character segmentation is the process where images of lines of text are broken down into individual characters. A key aspect of this process is the font used. Fonts can be broken into two general categories, based on the relative widths of characters. A fixed-pitch font is one where all characters occupy the same amount of space on the page. A proportional font is one where each character occupies a different amount of space depending on the width of the character. For example, a fixed width font would have characters “W” and “l” as occupying the same amount of space on the page, where a proportional font would allocate space proportionate to the size of the character. By

selecting fixed-pitched fonts, early systems could effectively hard code the expected size of characters. Modern systems cannot make this assumption and therefore have to employ more complex methods of isolating characters from each other. [15] In Tesseract, if text can be extracted with a reasonable confidence value, it is considered fixed pitch. Otherwise, text is assumed to be proportional and broken into characters based on the detected character cell size. [14]

The product of these preprocessing steps is to create a set of feature vectors with three dimensions- x-position, y-position, and direction. A length dimension is also calculated, but as a preprocessing step the vectors are broken down into a set of features of equal length. Once features are extracted, these attributes are measured to see which prototype features they are most similar to by using Euclidian distance. Characters consist of a cluster of features, and ultimately test features are assigned to a character using a K Nearest Neighbors, or KNN classification algorithm. KNN works by having a set of template features that have already been classified. The “nearest neighbors” or closest features to our test features, all have a class. A plurality vote is taken, and the test feature is assigned the class that gets the most votes. In this case, the classes that are assigned are characters in the English language. [18]

2.2) OCR Performance Considerations

There are two considerations in selecting an OCR solution. One is fraction performance and the other is general performance. Recipes often have fractions of a unit of measurement, such as a 1½ cups of milk or ¾ tablespoons sugar. Getting the quantities correct is important for calculating nutritional information, so OCR solutions for recognizing recipes must successfully

recognize different types of fractions. Part of the method is performing a test to assess performance for fractions.

For general performance, benchmarking studies show that both proprietary solutions outperform Tesseract. A 2021 study by Hegghammer show that Google and Amazon's OCR solutions generally have more accurate performance than Tesseract for larger text data sets. Hegghammer used social science documents as a test set and introduced various types of visual noise that might be encountered when photographing documents such as blur effects, ink stains, and scribbles. Document AI performed best, with Textract described as a "close second." [11] Both proprietary solutions are therefore preferable for higher performance than Tesseract.

2.3) Named Entity Recognition Definition

Named entity recognition (NER) is a natural language task which is necessary for our application. The premise of NER is that documents contain certain important entities that need to be recognized in an unstructured text, and linked to entries in a structured knowledge base. Often these are named entities such as people or places, but the entities can be anything that belongs to a given class or category. For our application, the named entities are food, units of measurement, and quantities of those units of measurement, as well as function words that add additional information. Detailed information about the tags applied is available in the appendix.

2.4) IOB Tagging

One common format for tagging in Named Entity Recognition is called IOB tagging. This format was established by Ramshaw and Marcus to make the boundaries of tagged entities less ambiguous. In this format, tags starting with a “B” mark the left-most beginning part of a chunk of words that belong to a single tag, or the ‘beginning’ of a tagged entity. Words that are part of the same tag that occur directly after the “B” are marked with an “I”, for ‘inside’. Words not in any tagged category are classed as “O” for ‘outside’. [35]

This is useful when two tagged entities are right next to each other. For example, a sentence reading “1 tsp of salt, baking soda, brown sugar” would have five tokens tagged as ingredients next to each other- “salt”, “baking”, “soda”, “brown” and “sugar”. Without IOB tagging, it would be difficult to reconstruct which adjacent words are part of the same tagged chunk. However, with IOB tagging, “salt”, “baking” and “brown” are all given B tags, and both “soda” and “sugar” would be given an I tag. This simplifies grouping by the parser.

2.5) Named Entity Recognition Performance

Performance for NER models are generally evaluated by calculating a F1 score, which itself is determined by using the harmonic mean of scores of precision and recall. There are four possible outcomes when a binary classification is given to an object by a model. A true positive (TP) is when a class is correctly assigned to a word by a model. A true negative (TN) is when the model correctly labels the word as not belonging to a class. A false positive (FP) is when the model incorrectly labels the word as belonging to a class when it does not. A false negative (FN) is when the model incorrectly labels a word as not belonging to a class, when it actually does.

Precision measures the number of relevant words among all words labeled as positive. Recall measures the number of relevant results out of all words that are actually belonging to the class, including false negatives. [25]

Precision and recall have an inverse relationship. A high recall can be trivially achieved by classing all words as positive, but this would result in poor precision. A high precision score can easily be achieved by having a very high threshold for labelling a word as positive, but this results in few words being labelled and result in a low recall. A strong model will have good performance scores for both precision and recall. The F1 score is the harmonic mean between these two scores, and therefore used for classifying accuracy in models. The full equations for all three measures can be viewed in Figure 2.2. [25]

$$Precision = \frac{T_p}{T_p + F_p}$$

$$Recall = \frac{T_p}{T_p + F_n}$$

$$F_1 = 2 * \frac{precision * recall}{precision + recall}$$

Figure 2.2 Formulae for precision and recall and F1 scores

2.6) Transformers

Natural language processing models have a variety of architectures, but recently a model called transformers has come to provide state-of-the-art performance for many natural language tasks, including NER. The General Language Understanding Evaluation (GLUE)

benchmark averages the performance of NER and language understanding across several test question sets. As of July 2022, six of the top ten highest performing models on GLUE are various implementations of this machine learning approach. [22] In this section, we will discuss the architecture of transformers.

The initial paper describing transformers was published by Vaswani et al working for Google in 2017. Transformers have two types of components in their architecture- encoders and decoders. There are multiple encoders and decoders in the model, contained in an encoder and decoder stack, respectively. Each encoder consists of a multi-headed self-attention mechanism and a position-wise feed-forward network. Decoders add an additional layer, which performs self-attention over the outputs provided from the encoder stack. We will describe each component of encoders, and then briefly discuss how decoders use these elements. [21]

The self-attention function first takes an input word vector, and calculates query, key and value vectors. These vectors are initialized randomly and adjusted using training data using residual dropout and label smoothing. Next, the function takes the dot product of the query and key vectors to calculate a score for each other word vector in the sentence. The next step is to divide the scores by the square root of the size of the key vector. This step is done because as the size of the key vector increases, the softmax function used in the next step produces very small gradients, reducing the effectiveness of the algorithm. Scaling the dot products results in more useful gradients. A softmax function takes the scores and converts them into a probability scaled from 0 to 1. Higher scores get a higher probability, and lower scores get lower probabilities. Next, the values vector gets scaled by the output of the softmax function, so that more important words get higher value scores, and less important ones get smaller values.

Finally, these weighted value vectors are summed up and turned into an output vector. The formula for this is expressed in figure 2.3. The multiheaded nature of the attention function means that the attention calculation is run multiple times. This gives us several differently initialized weights for query, key and value vectors. Once all of the different attention heads are calculated, they are concatenated and multiplied by a weight matrix, trained with the model, resulting in the output for the attention mechanism.

The feed-forward function is similar to convolutions in a very shallow neural network. According to Vashwani et al, "[The feed-forward function] consists of two linear transformations with a ReLU activation in between… Another way of describing this is as two convolutions with kernel size 1." The formula for this function can also be seen in Figure 2.3. After both the self-attention and feed-forward mechanism, there is a normalization step which serves to reduce training time. [26] Finally, the results of the encoder are fed forward to the next encoder in the stack as well as all decoders in the decoder stack.

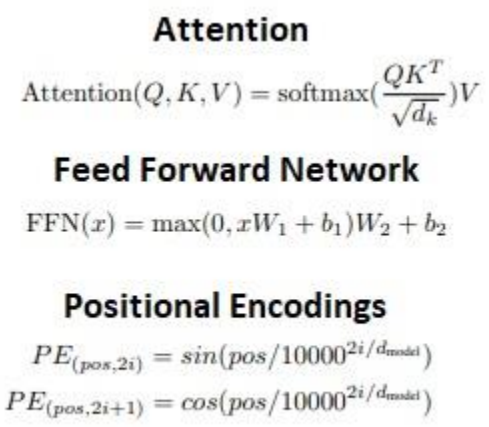

Figure 2.3 Formulae for transformers (Vashwani [26])

One more important element of this transformers model is positional encodings, which are added to the input word vectors in order to account for each word's position in a sentence. There are a variety of ways of doing positional encodings. The method used in the initial transformers paper is described in Figure 2.3. Vashwani et al state this method was chosen because "we hypothesized it would allow the model to easily learn to attend by relative positions, since for any fixed offset k, P Epos+k can be represented as a linear function of P Epos." In other words, each word in the model's sentence requires a relatively small amount of work to attend to another word in the sentence. [26]

2.7) BERT- Bidirectional Encoder Representation from Transformers

Transformers are the key component in a language model called BERT, which stands for Bidirectional Encoder Representations from Transformers. Many previous models to BERT attempted to predict output by reading in each word from either left-to-right or right-to-left. BERT instead considered the entire document at once, masking a percentage of random words and asking stacks of encoders and decoders to predict the output. The model was initially trained over a large corpus of data of hundreds of millions of words from Wikipedia and the BooksCorpus. The model can be fine-tuned on a smaller set of training data to perform a variety of NLP tasks, such as NER. This is done by adding another encoder layer with outputs corresponding to the various entities that need to be recognized. [27]

Food recognition is an emerging topic of NER research. Popovski et al conducted a survey of NER techniques in 2020 for food on a test set of 1000 manually annotated recipes. The best performing model by far was FoodIE. FoodIE was based on a more comprehensive NER

program called drNER which was designed to extract evidence-based nutritional impacts from scientific articles.[32] FoodIE was a simplified version of this designed to focus on food identification, using a manual ruleset and a complex set of handwritten rules to determine foods based on parts of speech from each sentence. FoodIE had a F1 rate of 96%. [23] However, transformer-based models were not tested as part of the survey. Stojanov et al used a transformer-based model in a 2021 paper, and their F1 performance on distinguishing food vs non-food had an F1 score ranging from 93-94%. [24]

2.8) FoodData Central

The United States Department of Agriculture maintains a database called FoodData Central, containing many common ingredients and their calorie information. [19] FoodData Central contains a variety of useful information that is used in the calorie counting application. An extensive list of ingredients is maintained in the database, and each ingredient is associated with a variety of nutrients including a metabolized calorie count calculated with Atwater General Factor System. [20]

FoodData Central supplies calorie counts per 100g of a given food for every ingredient. Frequently, alternative unit measurements are given for many ingredients. These include common quantity types such as unit count or volume. These alternative units have a grams per unit associated with them, which are also useful since most recipes do not contain measurement by grams. This is the basis for the database used by the application to ultimately calculate calorie values.

2.9) Vector Space and Word Embeddings

The FoodData Central database has thousands of items. In order to calculate the calories for an ingredient, for each candidate ingredient in a recipe, we must isolate the most similar ingredient type in FoodData Central to the entity string we have for our ingredient. Matching (or search) is an information retrieval task of finding the most relevant document for a given sample query.

Ingredients may not exactly match their FoodData Central counterpart, and simple similarity metrics are not always sufficient to bridge the gap. For example, "1 can of green beans" may be stored in the database as "Beans, snap, green, canned, regular pack, drained solids." [19] Simple string match metrics such as Levenshtein distance were tested over a small ten item data set and were not able to match several ingredients, including the green beans example above.

Many search algorithms used for information retrieval tasks make use of a vector space model, where, originally, each word in a vocabulary comprises a unique dimension in the space, and each term corresponds to a unique element in a vector. [37]

Vectors are objects with magnitude and direction, and can be thought of as rays or arrows pointing out in multi-dimensional space. In a vector space model, the first step to conducting search is to embed documents and the query as vectors, and the second step is to use a similarity metric between the query vector and each document vector. [31]

"Word embeddings" are a related but somewhat different concept to vector space. Embedding is a term used for designating a vector representation of a single word, where the values of the elements have been optimized to help distinguish the meaning of the word. There

are multiple ways of generating word embeddings, which can vary greatly depending on the model used to generate them.

2.10) Food2Vec Word Embeddings

Some of these embedding methods have been applied to the food domain. BERT generates embeddings, and those embeddings were considered for search. Published tools such as FoodNER [30] and BuTTer [29] provided code for creating BERT embeddings for food, but lacked a set of pretrained embeddings that could simply be used as part of a search tool.

Food2Vec is a publicly available set of word embeddings for food. The embeddings files were in JSON and publicly available on github. They were adapted with a python script to work with gensim for further extension as a search tool. Food2Vec also had the practical advantages of a pip repository in python, meaning that basic matching functions could be easily tested. Food2Vec embeddings were trained using Word2Vec training tasks over a collection of 95,896 recipes pulled from the Allrecipes.com website. [34]

2.11) Word2Vec Training Tasks

Word2vec trains its word embeddings on two related tasks. Word2Vec looks at a sliding window of words which we will say is size *N*. Half of the words in this window occur directly before our target word, and half of the words occur directly after it. The “Continuous Bag of Words” training method takes as an input the *N-1* words surrounding our target word and tries to guess our target word. The “skipgram” method flips the task, taking as an input the one ‘target’ word and trying to predict the surrounding context of *N-1* words.

Word2Vec also selects random words from our corpus that are not related and asks the model to predict that these words are in fact unrelated, in a process called “negative sampling.” Word2Vec conducts its training by attempting either of these two tasks over a corpus. First, the model’s initial values are randomized. The model guesses the target word or words based on the task, then checks to see what the actual value of the word or words is. It calculates an error percentage based on how correct or incorrect the guesses were, and adjusts the embeddings of each word in the task accordingly. It does this words that are correct, and conducts negative sampling to train other words as incorrect. [33]

2.12) Cosine Similarity

Once documents are embedded, we have to calculate a similarity metric between our query embedding and the document embeddings for the corpus. Distance between two documents seems like an obvious choice for a similarity metric, but distance is susceptible to being strongly influenced by the size of documents. Instead, a metric called cosine similarity is more often used, which measures the differences in angle between where two vectors are pointing. Calculating the cosine similarity is simply a matter of summing up the dot product between each element in the two vectors. For document D and query Q, figure 2.4 shows the cosine similarity formula. *W_tiQ* represents the *i*th term in the query, and *W_tiD* represents the *i*th term in the document.

$$Sim(\vec{D}, \vec{Q}) = \sum_{t_i \in Q, D} w_{t_{iQ}} \cdot w_{t_{iD}}$$

Figure 2.4 Formula for cosine similarity (Singhal [31])

3) Method

3.1) Outline of Application Architecture

The application's architecture starts with the input- an image of a recipe. Next, the Textract OCR engine is used to obtain a text version of the recipe. The application then uses a BERT NER model to label important sections of text, such as ingredients, quantities and units. These sections of text are then used to search a database with food calorie values called FoodData Central. Pretrained word embeddings from Food2Vec are used to embed the ingredient and the database descriptions in vector space. Cosine similarity is then calculated to find the closest matching ingredients and database entries. A logarithmic weight based on ingredient counts over our corpus is multiplied with the cosine similarity score. The highest result is pulled from the database and calories are calculated for the ingredient. The final calorie output is a sum of the calorie values of each ingredient.

Evaluation of the results will be done by running 5 recipes from the corpus through the pipeline. Calories will also be manually tabulated for those recipes. The amount of difference between the output of the calorie counting pipeline and the manually tabulated calorie count is the percentage of error. The percentage of error will be averaged to come up with an overall accuracy for the application.

A detailed outline of each step in the pipeline continues below.

3.2) Corpus Data

Test data is used to train and assess the quality of a number of functions in the pipeline. All test recipes were pulled from a corpus of 19 cookbooks from the Cookbook and Home Economics Collection were used from archive.org. [28]

3.3) Selecting an OCR Application

The first step was to assess the accuracy of various OCR engines on a crucial aspect of recipes, which is the recognition of fractions. To evaluate OCR functionality, a test set of four pages from three different cookbooks from the corpus were used as input to three OCR engines- Tesseract, Document AI and Textract. Pages with a variety of fraction types on them were selected, so quantities would not just consist of one fraction like ½ or ¼. In the test set, a total of 38 fractions were spread over the four pages, with 23 of them being ½ and 15 being other fraction types. The accuracy of the OCR solution was manually determined. After testing multiple applications, Amazon Textract was determined to be the most accurate and selected for the application.

3.4) NER Model

Once a text version of the recipe is obtained, the remaining steps to computing calories are to analyze the text and categorize all of the necessary information to calculate the calorie content. A BERT model was trained specifically for this task. Pages with recipes from the baking sections were used to create training data. (It was noted that every one of the cookbooks selected had a baking selection.) The images were run through Amazon's Textract OCR engine and then manually marked up using the BIO schema with the tags specified in table 3.1.

Tag	Definition
ING	Name of ingredient
UNIT	Type of measurement
QTY	Number indicating amount of unit
CONT	Countable ingredients lacking units
NQT	Ingredients with no quantity specified
FUNC	Indicates parser should function differently based on word

Table 3.1 Tags and Definitions (Details in Appendix)

The coding was done manually by the researcher. Additional information about the classification scheme is available in appendix A. One hundred thousand words were included in the training set. The training data was split into train and test sets. 90% of the inputs were set as training set and the remaining 10% was test set. Training data was iterated over 10 epochs.

3.5) Parsing Queries

In order to calculate calories, ingredients, units and quantities all need to be present. The parser groups tagged words together into complete sets called queries. The process is documented in figure 3.1. If any information is missing from a query, the calorie calculation cannot be performed. The parser takes several passes to create a query. As a first pass, the parser strips out all O-tags generated by the NER. The parser detects division symbols and ensures they are grouped together with a numerator and denominator to make a fraction.

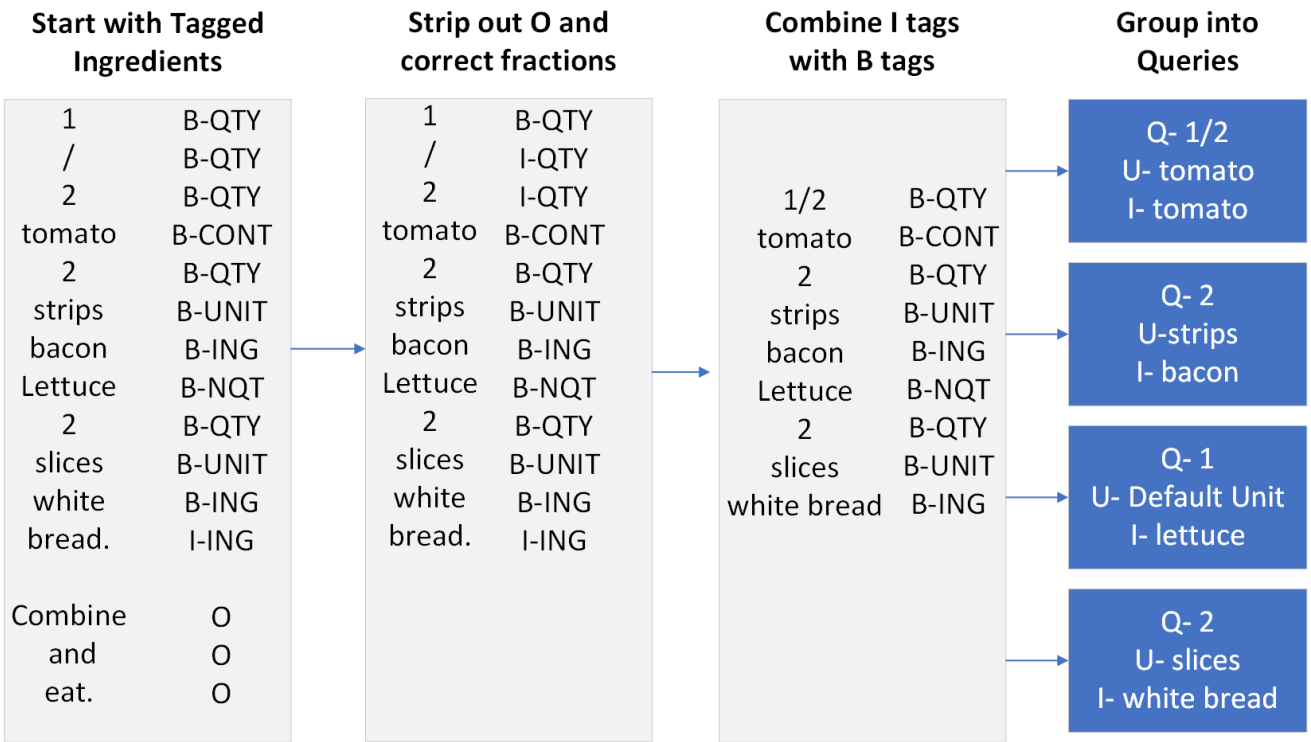

Figure 3.1 Parser Workflow

The parser goes through the tokens backwards, combining I-tags with the B-tags that are to their left in the original document.

Query sets are formed next, which require ingredient, unit and quantity information. New queries are created every time a QTY or NQT tag is encountered. NQT ingredients are given one default serving unit. UNIT and ING tags are put in query sets as they are encountered. CONT tags are classified as both unit and ingredient elements of the query, since they represent both a unit and ingredient. For complex queries involving FUNC, any entities following the word "or" are ignored, since only the first option is selected. Entities following the word "and" are given copies of the quantity and unit information from the previous query. All queries are then given a final pass and any queries missing an ingredient name or quantity are discarded.

3.6) Search

Once the data has been formulated as queries, the model must match each query to an entity in FoodData Central. First, the model embeds the ingredient information from the query in vector space. Next, the model calculates the cosine similarity between the ingredient and all the descriptions of food in our database, which includes data from FoodData Central, modified descriptions, weights, and additional entries to make up for gaps in FoodData Central's database.

Weights are applied to the cosine similarity scores. The weights are softmax scores, calculated by taking the log of the number of times an ingredient occurred in the corpus. Each ingredient's softmax score is multiplied by the cosine similarity score to get the final weighted similarity metric. The top scoring ingredient's calorie information is pulled from the database. Unit information is calculated based on the nearest string match.

3.7) Weighting

The weights used as part of the search process were derived by running the previously trained ingredient recognition BERT model over a 400,000 token selection from the archive.org collection of cookbooks. There were found to be 29,543 occurrences of ingredients and 1342 unique ingredients in this selection. As figure 3.2 shows, certain ingredients were much more likely than others to be in a given recipe. The top 25 ingredients made up 54.6% of all occurrences of ingredients in a recipe. The top 100 made up 76% of all occurrences of ingredients. Applying the weighting function requires manually matching the search result, so only the top 100 most common unique ingredients had their softmax scores set. The remaining unique ingredients in the database were set to a softmax score equivalent of 6 occurrences, because that is approximately equal to the 6794 remaining ingredient instances spread evenly over the remaining 1242 ingredients.

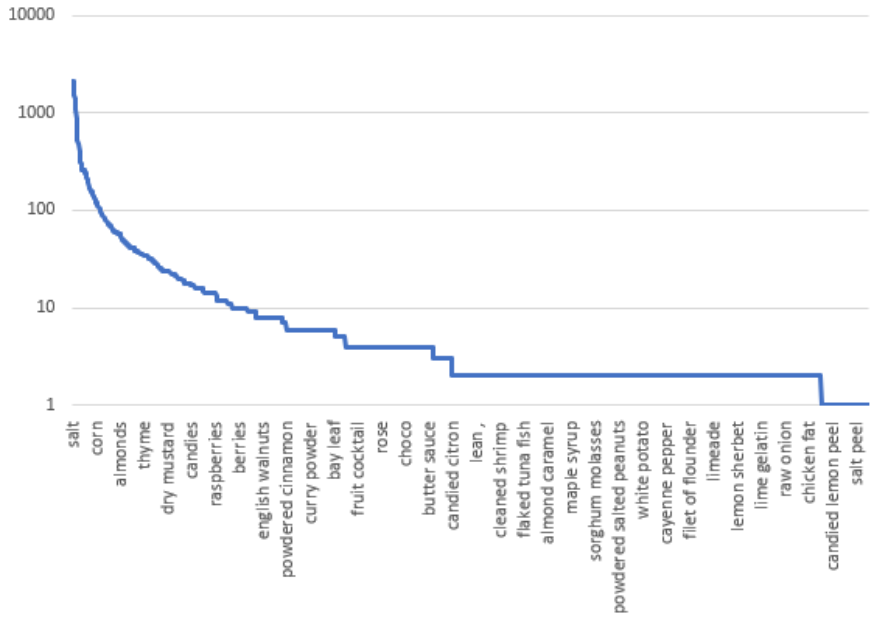

Figure 3.2. Ingredient distribution over corpus

3.8) Calculating Calories

The number of calories in a given recipe is calculated as the sum of the calories of each ingredient contained in the recipe. For a given ingredient *i,* we calculate the calories multiplying together three numbers- *g_i*, *u_i* and *q_i*. The quantity of units of measurement in the ingredient in the recipe is represented by *q_i*. The number of calories per gram of a given ingredient is *g_i*. The number of grams per unit of measurement in the recipe for the ingredient is represented by *u_i*. *q_i* is taken from the query, and the other two quantities are taken from the ingredient and unit matched in FoodData Central. Figure 3.3 shows an example calculation of a single ingredient's calories.

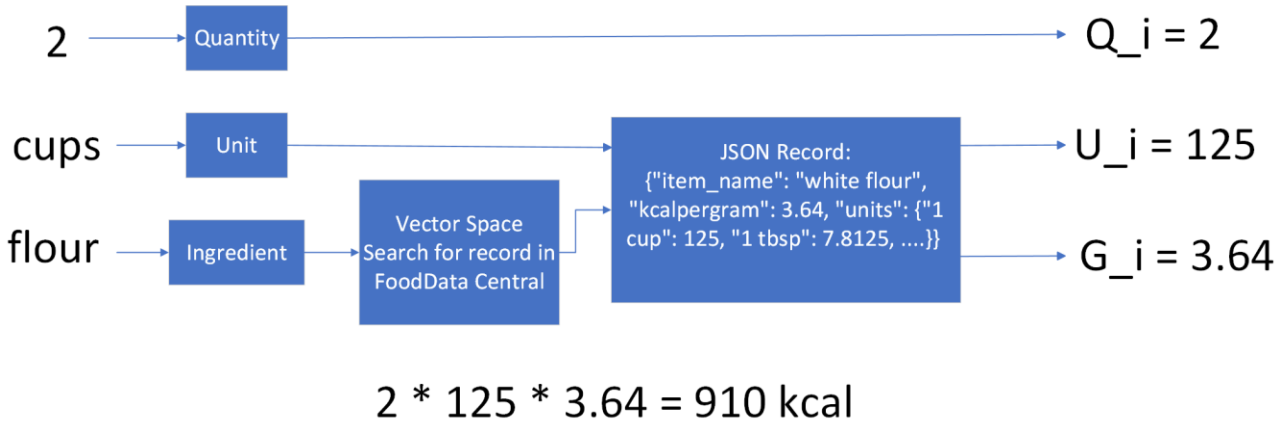

Figure 3.3 Calculating calories for a single ingredient

To obtain a calorie count of the entire recipe, the calorie count application calculates the calorie counts of all ingredients and sums them together. In a recipe with *j* as the total number of ingredients, the expression we will use to ultimately calculate a recipe's calorie value is represented in figure 3.4. Once calculated, we divide by the number of servings *s* to get a calories per serving count.

$$\frac{\sum_i^j g_i u_i q_i}{s}$$

Figure 3.4 Equation for calories per serving for a recipe

4) Results

4.1) OCR Fraction Performance

	½	¼	1/8	¾	2/3	1/3	Total	% Accuracy
Ground Truth	23	9	3	1	1	1	38	1.00
Tesseract	0	0	0	0	0	1	1	0.03
Amazon Textract	18	9	3	1	0	1	32	0.84
Google Document AI	22	5	0	1	0	1	29	0.76

Table 4.1 OCR engine performance for fractions

Table 4.1 demonstrates the overall results of the fraction OCR on the test set. Tesseract's accuracy was only 3%, making it unacceptable as a solution. Google Document AI had 76% accuracy. Textract proved the best at recognizing fractions with an overall accuracy of 84%. Moreover, once automated corrections were performed to split the double-digit numerators, this accuracy increased to 92%. Because Textract had the highest accuracy for fractions, it was selected at the top candidate for the project's OCR component.

4.2) NER Model Performance

The calorie counter app F1 score was 96.4%. Accuracy was 99.6%. Precision was 96.0%, and the recall was 96.7%. Validation loss decreased for both training and test data. Figure 4.1 displays the training loss decreasing over 10 epochs.

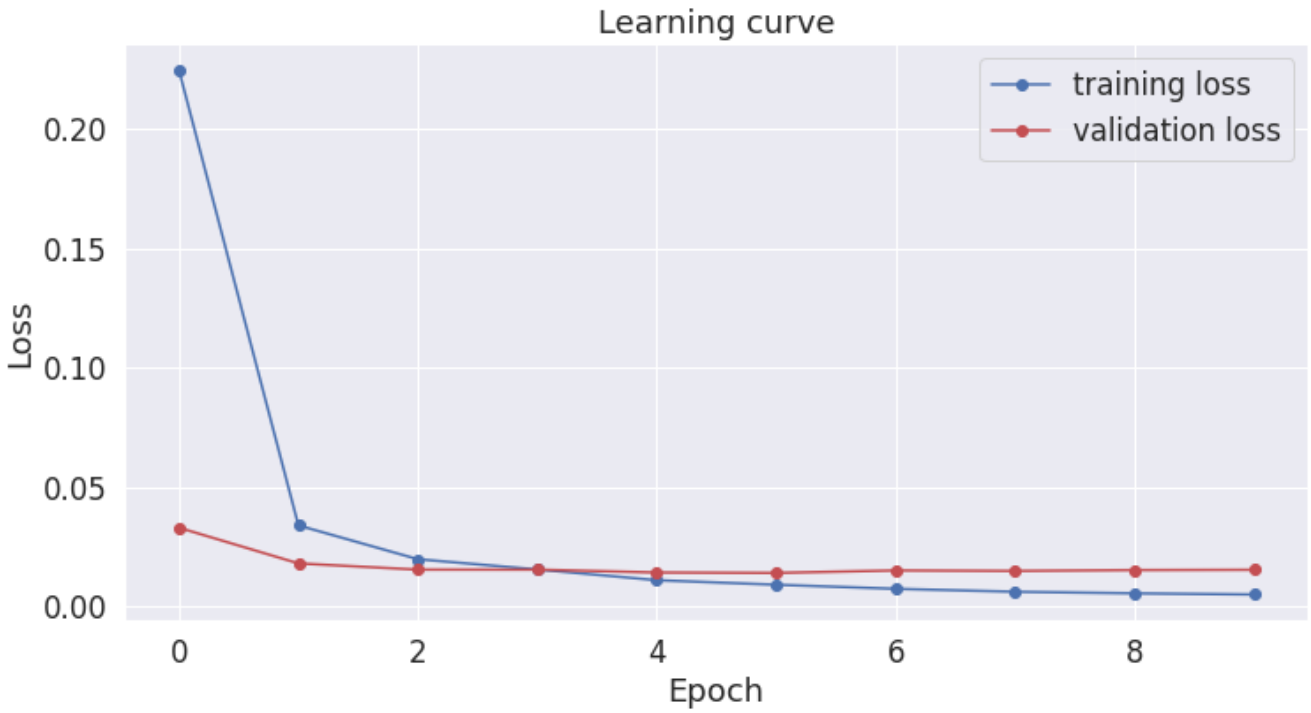

Figure 4.1 BERT model training loss

4.3) Search

To test ingredient search, the top 100 most common ingredients were formatted as queries and searched using the weighted cosine similarity metric. Of these ingredients, 76 directly matched the term, 18 incorrectly matched with a different ingredient, and 6 were incorrectly chunked partial ingredients (such as “baking”, “green” or “grated”). Disregarding the incorrectly formatted ingredients, the success rate for searching correctly formatted ingredients was 80.9% for the top 100.

The top 25 most common ingredients had a higher success rate. Of these ingredients, 23 directly matched the term, 1 incorrectly matched and 1 was incorrectly labelled (“baking”). This would make for a 92% success rate over the most common ingredients.

4.4) Full Pipeline

Five actual images of recipes were also tested on the entire pipeline. Full calorie counts were not provided for the recipes in the corpus and as such were not able to be assessed. However, the results of each ingredient search were manually reviewed. 37 ingredients were successfully matched, and 2 ingredients were not correctly matched. This was an ingredient matching success rate of 94.9%. Calorie results were manually calculated and compared to the calories suggested by the pipeline. The calorie counts were 96.5% accurate.

5) Discussion

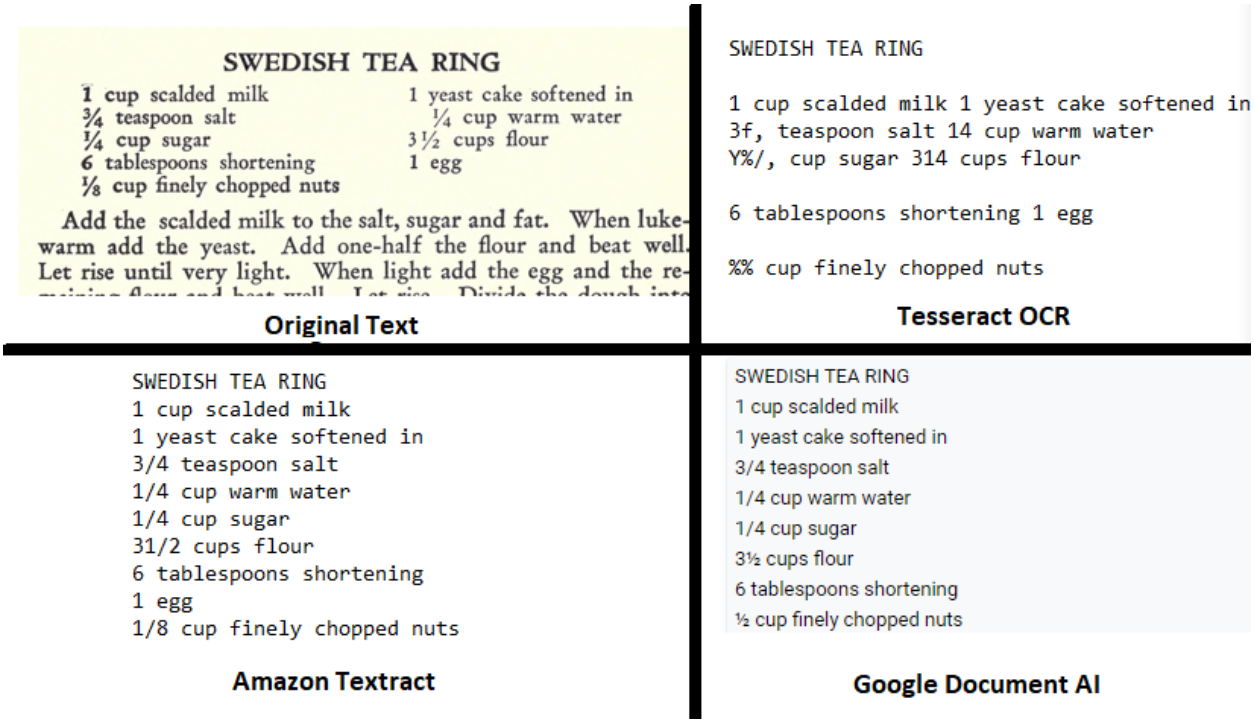

Figure 5.1 Comparison of one recipe over multiple OCR solutions.

5.1) OCR Performance on Fractions

Based on manual inspection, testing revealed several types of errors. Tesseract failed outright to recognize any individual fraction characters, such as ¼, with the only success being a fraction that was not a special character (e.g., “1 / 4”). Proprietary OCR solutions performed much better, but not perfectly. Figure 5.1 (page 28) demonstrates how each OCR solution performed on a single recipe, and also shows the two types of errors that occurred, namely, incorrect denominators and incorrect mixed fractions.

Incorrect denominators were more common in Document AI, where most fractions were recognized as ½, regardless of their actual denominator. ½ was the most frequent fraction in the test set, so Document AI still managed to be relatively accurate overall. Textract performed much better at recognizing denominators and numerators, missing only one out of the fifteen non-½ fractions. This type of error is difficult to sanity check, as it is feasible that a recipe would have either ¼ cup or ½ cup of a given ingredient. Figure 5.1 (page 28) demonstrates how 1/8 cup chopped nuts in the original recipe becomes ½ cup chopped nuts in Document AI.

Incorrect mixed fractions were also an issue. Incorrect mixed fractions occurred when numbers such as 1½ had whole number components that were concatenated with the fraction, so 1½ might be recognized as 11/2. Document AI often made the distinction between fractions and whole numbers, but Textract did not perform as well in recognizing the mixed nature of the fraction. For example, in Figure 4.1 we can see that Textract successfully recognized each number in the fraction, but did not put a space between the whole number and the numerator of the fraction, resulting in the double-digit numerator 31/2 instead of 3½. This is a problem,

but not a severe one, as it is relatively easy to detect and correct during post-processing. Most recipes will more be more likely to call for ingredients with denominators of 2, 3, 4 or 8, where a double-digit numerator would be noticeably wrong. The only double-digit denominator likely to be used is 16, but it would be unlikely for a recipe to call for 11/16ths of an ingredient, rather than some easier to measure but roughly equal number like 2/3rds.

Textract was the highest performing model, but it also seems likely that Document AI and Tesseract were either not trained on fractions altogether or only trained to recognize ½. Further work could be done to improve fraction performance by including more fractions in training data for OCR models.

5.2) NER Model Performance

For the custom BERT model, overall results were in line with other models. As discussed in the background section, FoodIE had a F1 rate of 96% and Stojanov et al had a transformer-based model with a F1 score ranging from 93-94%.[23] [24]

While the calorie counter app has a slightly higher F1 score than the other applications, it should be noted that the calorie counter application labelled quantities and units, not just food. These tasks were likely more straightforward than labelling food, which could be thousands of possible words. For units, there's a much smaller selection of a few dozen words that are typically used. Quantities are usually numbers, either full integers or fractions. These more straightforward tasks may have improved the scoring of the labelling overall.

These results also show that the labelling non-ingredient data can also be done at a high level of accuracy. However, there is a limitation regarding having only one person code the data that was used to train the NER model. This limitation does not allow this thesis to quantify how

difficult or ambiguous this coding task is or to assess inter-rater agreement. These aspects were considered beyond the scope of the thesis.

5.3) Search

Search is the worst performing component with an 80.9% success rating for the 100 most frequently occurring ingredients. The embeddings used for search were domain specific but did not match well without the weighting component. Perhaps a more purpose-built set of embeddings could improve the match rate. Or perhaps the NER model could be trained to directly match ingredients with their respective database entry, avoiding the step of search altogether. While disappointing, the top 25 ingredients had a higher success rate of 92%, and due to the top-heavy distribution of ingredients, these inaccuracies had less impact on the overall results than might be expected.

5.4) Full Pipeline Test on Recipes

Running five test recipes through the entire pipeline yielded a relatively successful 94.8% ingredient match rate and 96.5% calorie match rate, above the 85% baseline established. This demonstrates that the pipeline is feasible. If implemented in a phone application, it could be used as a way to improve the speed and accuracy of dietary self-monitoring for people using cookbooks.

5.5) Conclusion

Most components of the application were more successful than the 85% benchmark set by other calorie counting applications. The OCR application had an accuracy rating of 92% with cleanup. OCR could likely be improved further by training fractions more thoroughly. The NER model had a 96.4% F1 score, on par with many other food NER tools. Search was near the

baseline at 84%. Improved embeddings for search are an area for further research, which may improve the success rate even further. Overall, the prototype pipeline for the calorie counting app has met the baseline set for general feasibility.

Bibliography/Works Cited/References

[1] Collier, R. (2011). US obesity rates growing [Article]. *Canadian Medical Association Journal (CMAJ)*, *183*(11), E723. https://doi.org/10.1503/cmaj.109-3936

[2] Long, D. A., Reed, R., & Lehman, G. (2006). The Cost of Lifestyle Health Risks:: Obesity [Article]. *Journal of Occupational and Environmental Medicine*, *48*(3), 244–251. https://doi.org/10.1097/01.jom.0000201568.73562.a2

[3] *The practical guide : identification, evaluation, and treatment of overweight and obesity in adults*. (2000). [Book]. National Heart, Lung, and Blood Institute.

[4] Hutchesson, M. J., Rollo, M. E., Callister, R., & Collins, C. E. (2015). Self-Monitoring of Dietary Intake by Young Women: Online Food Records Completed on Computer or Smartphone Are as Accurate as Paper-Based Food Records but More Acceptable. Journal of the Academy of Nutrition and Dietetics, 115(1), 87–94. https://doi.org/10.1016/j.jand.2014.07.036

[5] Burke, L. E., Wang, J., & Sevick, M. A. (2011). Self-Monitoring in Weight Loss: A Systematic Review of the Literature. Journal of the American Dietetic Association, 111(1), 92–102. https://doi.org/10.1016/j.jada.2010.10.008

[6] White, Martha. Recipe for Success: Cookbook sales survive shift to digital media. NBC News. https://www.nbcnews.com/business/consumer/recipe-success-cookbook-sales-survive-shift-digital-media-n900621

[7] Dunn, C. G., Turner-McGrievy, G. M., Wilcox, S., & Hutto, B. (2019). Dietary Self-Monitoring Through Calorie Tracking but Not Through a Digital Photography App Is Associated with Significant Weight Loss: The 2SMART Pilot Study—A 6-Month Randomized Trial. Journal of the Academy of Nutrition and Dietetics, 119(9), 1525–1532. https://doi.org/10.1016/j.jand.2019.03.013

[8] Burke L.E., Conroy M.B., Sereika S.M., et al: The effect of electronic self-monitoring on weight loss and dietary intake: a randomized behavioral weight loss trial. Obesity (Silver Spring) 2011; 19: pp. 338-344

[9] Coulston, A. M., Rock, C., & Monsen, E. R. (2001). Nutrition in the prevention and treatment of disease / edited by Ann M. Coulston, Cheryl L. Rock, and Elaine R. Monsen. Academic Press.

[10] Vincent, Luc. “Announcing Tesseract OCR.” August 30, 2006. From website: http://googlecode.blogspot.com/2006/08/announcing-tesseract-ocr.html

[11] Hegghammer, Thomas. "OCR with Tesseract, Amazon Textract, and Google Document AI: a benchmarking experiment." Journal of Computational Social Science (2021): 1-22.

[12] Amazon.com. “AWS Announces General Availability of Amazon Textract.” *Amazon.com, Inc. - Press Room,* https://press.aboutamazon.com/news-releases/news-release-details/aws-announces-general-availability-amazon-textract.

[13] Liu, Lewis, and Yang Liang . “Introducing Document AI Platform, a Unified Console for Document Processing.” Google Cloud Blog, https://cloud.google.com/blog/products/ai-machine-learning/google-cloud-announces-document-ai-platform.

[14] Smith, Ray. "An overview of the Tesseract OCR engine." *Ninth international conference on document analysis and recognition (ICDAR 2007)*. Vol. 2. IEEE, 2007.

[15] Lu. (1995). Machine printed character segmentation —; An overview. Pattern Recognition., 28(1), 67–80. https://doi.org/10.1016/0031-3203(94)00068-W

[16] Garg, Naresh, and N. Garg. "Binarization techniques used for grey scale images." *International Journal of Computer Applications* 71.1 (2013): 8-11.

[17] History of the Tesseract OCR engine: what worked and what didn't

[18] Akhil, S. "An overview of tesseract OCR engine." *A seminar report. Department of Computer Science and Engineering National Institute of Technology, Calicut Monsoon*. 2016.

[19] U.S. Department of Agriculture, Agricultural Research Service. FoodData Central, 2019. fdc.nal.usda.gov.

[20] Mayxard, L. A. "The Atwater system of calculating the caloric value of diets." *Journal of Nutrition* 28 (1944): 443-452.

[21] Vaswani, Ashish, et al. "Attention is all you need." Advances in neural information processing systems 30 (2017).

[22] Wang, Alex, et al. "GLUE: A multi-task benchmark and analysis platform for natural language understanding." *arXiv preprint arXiv:1804.07461* (2018).

[23] Popovski, Gorjan, Barbara Koroušić Seljak, and Tome Eftimov. "A survey of named-entity recognition methods for food information extraction." *IEEE Access* 8 (2020): 31586-31594.

[24] Stojanov, Riste, et al. "A Fine-Tuned Bidirectional Encoder Representations From Transformers Model for Food Named-Entity Recognition: Algorithm Development and Validation." *Journal of Medical Internet Research* 23.8 (2021): e28229.

[25] Idris, Ivan. *Python data analysis cookbook*. Packt Publishing Ltd, 2016.

[26] Ba, Jimmy Lei, Jamie Ryan Kiros, and Geoffrey E. Hinton. "Layer normalization." *arXiv preprint arXiv:1607.06450* (2016).

[27] Devlin, Jacob, et al. "Bert: Pre-training of deep bidirectional transformers for language understanding." *arXiv preprint arXiv:1810.04805* (2018).

[28] "Cookbooks and Home Economics." *Cookbooks and Home Economics, Internet Archive*, https://archive.org/details/cbk.

[29] Cenikj, Gjorgjina, et al. "BuTTER: BidirecTional LSTM for Food Named-Entity Recognition." *2020 IEEE International Conference on Big Data (Big Data)*. IEEE, 2020.

[30] Stojanov R, Popovski G, Cenikj G, Koroušić Seljak B, Eftimov T. A Fine-Tuned Bidirectional Encoder Representations From Transformers Model for Food Named-Entity Recognition: Algorithm Development and Validation. *J Med Internet Res*. 2021;23(8):e28229. Published 2021 Aug 9. doi:10.2196/28229

[31] Singhal, Amit (2001). "Modern Information Retrieval: A Brief Overview". Bulletin of the IEEE Computer Society Technical Committee on Data Engineering 24 (4): 35–43.

[32] Eftimov T, Korousˇić Seljak B, Korosˇec P (2017) A rule-based named-entity recognition method for knowledge extraction of evidencebased dietary recommendations. PLoS ONE 12(6): e0179488. https://doi.org/10.1371/journal. pone.0179488

[33] Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013a). Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781.

[34] Altosaar, Jaan. *food2vec - Augmented Cooking with Machine Intelligence*, 20 Aug. 2018, https://jaan.io/food2vec-augmented-cooking-machine-intelligence/. Accessed 3 Aug. 2022.

[35] Ramshaw, Lance A., and Mitchell P. Marcus. "Text chunking using transformation-based learning." *Natural language processing using very large corpora*. Springer, Dordrecht, 1999. 157-176.

[36] Attokaren, David J., et al. "Food classification from images using convolutional neural networks." *TENCON 2017-2017 IEEE Region 10 Conference*. IEEE, 2017.

[37] Salton, Gerard, Anita Wong, and Chung-Shu Yang. "A vector space model for automatic indexing." *Communications of the ACM* 18.11 (1975): 613-620.

[38] Okamoto, Koichi, and Keiji Yanai. "An automatic calorie estimation system of food images on a smartphone." *Proceedings of the 2nd International Workshop on Multimedia Assisted Dietary Management*. 2016.

[39] Zhang, Weiyu, et al. ""snap-n-eat" food recognition and nutrition estimation on a smartphone." *Journal of diabetes science and technology* 9.3 (2015): 525-533.

[40] Rhyner, D.; Loher, H.; Dehais, J.; Anthimopoulos, M.; Shevchik, S.; Botwey, R.H.; Duke, D.; Stettler, C.; Diem, P.; Mougiakakou, S. Carbohydrate estimation by a mobile phone-based system versus self-estimations of individuals with type 1 diabetes mellitus: A comparative study. J. Med. Int. Res. 2016, 18, e101. [CrossRef] [PubMed]

[41] Poply, Parth, and J. Angel Arul Jothi. "Refined image segmentation for calorie estimation of multiple-dish food items." *2021 International Conference on Computing, Communication, and Intelligent Systems (ICCCIS)*. IEEE, 2021.

Appendix) Tag Classification Scheme

Three pieces of information are required to calculate calories, and the basic tags in the classification scheme correspond to these. The ING tag indicates the name of an ingredient, which can be used to look up the nutritional information in food central. The UNIT tag specifies the unit type, a tablespoon or gallon for example. The quantity of units of measurement in the ingredient is captured by a quantity tag QTY. Those are the most basic tags.

However, recipes were often inconsistently formatted in the training set, which required additions to the simple schema above to account for potentially missing information. These special categories account for various special cases that exclude or duplicate certain elements from the three basic categories. Such special categories include function words, unquantified ingredients, and countable ingredients.

Function words are labelled FUNC, and are words such as “and”, “or” and “of” that often served to group ingredients together, provide an option between two ingredients. “And” means that the unit and quantity from the previous ingredient should be applied to the next ingredient. For example, a recipe may call for “1 tbsp cinnamon, cloves, and nutmeg.” The quantity 1 tbsp applies to each ingredient. “Or” is also a function word that suggests alternatives. When a recipe calls for something like “2 tbsp honey or 1 tbsp sugar.”, only the first option in groups with “or” is calculated.

Unquantified ingredients often occurred in recipes, and are labelled NQT, for not quantified. This meant that ingredients were listed without any specific quantity or unit associated. Very frequently, these were spices. Many recipes simply called for “salt and pepper” and not specify any particular quantity or unit, presumably to taste. Other examples might

include “eggs, breadcrumbs and oil” for frying, or “broth to cover” certain recipes. The FoodCentral database includes default values for each ingredient which represent a common amount for a single serving. In the case of unquantified ingredients, such as “broth to cover”, the default values from the FoodCentral database are used for calculations.

Countable ingredients labeled CONT are also a unique case which requires special consideration. For foods which come in discrete units, the food may or may not have a unit explicitly listed. For example, a recipe may call “3 cups of chopped carrots” or simply “3 chopped carrots.” In the second case, a carrot is both the unit and the ingredient. The parser considers these countable ingredients as both a unit and an ingredient.