Inverse Cooking: Recipe Generation from Food Images

Vasundhara Venkata Krishna
CIDSE
Arizona State University
Tempe, Arizona
vvvenka1@asu.edu
ASURITE: 1218418357

Arkan Abuyazid
CIDSE
Arizona State University
Tempe, Arizona
aabuayz1@asu.edu
ASURITE: 1219636340

Naga Sai Aishwarya Tallapragada
CIDSE
Arizona State University
Tempe, Arizona
ntallapr@asu.edu
ASURITE: 1218515961

***Abstract*—Food recognition challenges current computer vision systems to go beyond the merely visible. When compared with natural image understanding, visual ingredient prediction requires high-level reasoning and prior knowledge. This poses additional challenges, as food components have high intra-class variability, heavy deformations occur during cooking, and ingredients are frequently occluded in a cooked dish. Generally, the problem has been formulated as a retrieval task, where a recipe is retrieved from a fixed dataset based on the image similarity score. But these systems fail when a matching recipe for the image query does not exist in the static dataset. We propose to study an inverse cooking system that given images of prepared dishes, recreates the respective recipes. This system will predict ingredients as sets by modeling their dependencies without imposing any order. Then, the system will generate instructions conditioned on images and inferred ingredients. Finally, we will demonstrate the superiority of the system against state-of-the-art image-to-recipe retrieval approaches.**

***Index Terms*—Computer Vision systems, food, food recognition, prediction, dataset**

I. Introduction

The sharing of food has brought people together since the beginning of time. It's how we make friends, nurture relationships, celebrate milestones, mend conflicts and feel gratitude for life. Food helps us create memories. People enjoy food photography because they appreciate food. Behind each meal there is a story described in a complex recipe. Recreating those recipe for our friends or family is a way to re-live those memories. In the past, food was mostly prepared at home, but nowadays we frequently consume food prepared by third parties (e.g. takeaways, catering and restaurants). Thus, the access to detailed information about prepared food is limited and, as a consequence, it is hard to know precisely what we eat. Therefore, we argue that there is a need for inverse cooking systems, which are able to infer ingredients and cooking instructions from a prepared meal.

Traditionally, the image-to-recipe problem has been formulated as a retrieval task, where a recipe is retrieved from a fixed dataset based on the image similarity score in an embedding space. The performance of such systems highly depends on the dataset size and diversity, as well as on the quality of the learned embedding. Not surprisingly, these systems fail when a matching recipe for the image query does not exist in the static dataset. An alternative to overcome the dataset constraints of retrieval systems is to formulate the image-to-recipe problem as a conditional generation one. Therefore, in this paper, we present a system that generates a cooking recipe containing a title, ingredients and cooking instructions directly from an image. The system first predicts ingredients from an image and then conditions on both the image and the ingredients to generate the cooking instructions.

The main scope of this project is Computer Vision and Image analysis. Image analysis is the extraction of meaningful information from images; mainly from digital images by means of digital image processing techniques. Image analysis tasks can be as simple as reading bar coded tags or as sophisticated as identifying a person from their face. Computer vision is a field of artificial intelligence that trains computers to interpret and understand the visual world. Using digital images from cameras and videos and deep learning models, machines can accurately identify and classify objects — and then react to what they "see." Computer vision seeks to understand and automate tasks that the human visual system can do, and a bit more!

In this paper we study the instruction generation problem as a sequence generation conditioned on the two methods image analysis and its predicted ingredients. We extensively evaluate this system on the large-scale Recipe1M dataset that contains images, ingredients and cooking instructions, showing satisfactory results.

II. Methods

Generating a recipe (title, ingredients and instructions) from an image is a challenging task. It requires an understanding of the ingredients composing the dish and the transformations the ingredients went through, e.g. slicing, blending or mixing with other ingredients. Instead of obtaining the recipe from an image directly, the recipe generator would work better if predicting the ingredients list is added as an intermediate step. The sequence of instructions would then be generated conditioned on both the image and its corresponding list of ingredients. The interplay between image and ingredients could provide additional insights on how the ingredients were processed to produce the resulting dish. The recipe generation

system takes a food image as an input and outputs a sequence of cooking instructions, which are generated by means of an instruction decoder that takes as input two embeddings. The first one represents visual features extracted from an image, while the second one encodes the ingredients extracted from the image. The Recipe Generator Model can be visualized as shown in Figure 1.

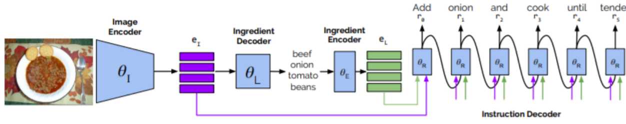

Fig. 1. Recipe Generation Model

The Recipe Generator is a transformer based model and it includes the following components:

- *Image Encoder*.
  The image encoder is parametrized by θ_I and used for extracting the visual features e_I of an image. This process is called as image feature extraction.
- *Ingredient Decoder*.
  Ingredient decoder is parametrized by θ_L. This decoder takes the image features e_I as input and predicts the ingredients.
- *Ingredient Encoder*.
  The predicted ingredients from the ingredient decoder are encoded into ingredient embeddings e_L. This Ingredient Encoder is parametrized with θ_e.
- *Instruction Decoder*.
  The Instruction Decoder is parametrized with θ_R. It takes image embeddings e_I, ingredients embeddings e_L and previously predicted words $(r_1, r_2, ...r_{t-1})$ as input and produces a sequence of cooking instructions $R = (r_1, r_2, ...r_T)$ where r_t denotes a word in the current sequence.

The output of the recipe generator is a cooking recipe containing a title, ingredients and cooking instructions directly from an image. An example of the output is as given in Figure 2.

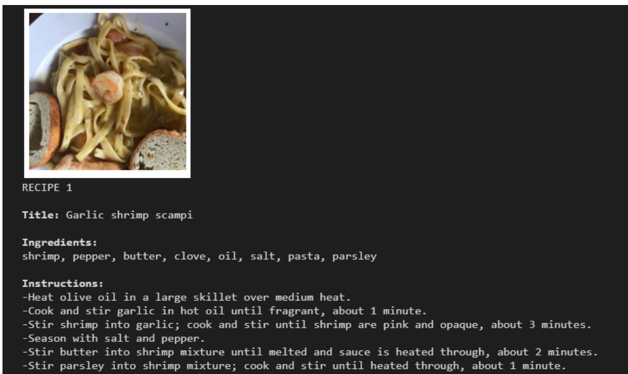

Fig. 2. Example output of Recipe Generator

A. Recipe Generation Model

Understanding the instruction decoder and ingredient decoder in depth.

- *Cooking Instruction Transformer(Instruction Decoder)*. The sequence of instructions $r_1, r_2, ...r_T$ is produced by with the help of an instruction transformer. The title is predicted as the first instruction. This transformer is conditioned jointly on two inputs: the image representation e_I and the ingredient embedding e_L. The instruction decoder is composed of transformer blocks, each of them containing two attention layers followed by a linear layer. The first attention layer applies self-attention over previously generated outputs, whereas the second one attends to the model conditioning in order to refine the self-attention output. The three different fusion strategies are:
  1) *Concatenated Attention*. This strategy first concatenates both image e_I and ingredients e_L embeddings over the first dimension $e_{concat} \epsilon R^{(K+P)d_e}$. Then, attention is applied over the combined embeddings.
  2) *Independent Attention*. This strategy incorporates two attention layers to deal with the bi-modal conditioning. In this case, one layer attends over the image embedding e_I , whereas the other attends over the ingredient embeddings e_L. The output of both attention layers is combined via summation operation.
  3) *Sequential Attention*. This strategy sequentially attends over the two conditioning modalities. In this design, the following orderings are considered: 1st image first where the attention is first computed over image embeddings e_I and then over ingredient embeddings e_L; 2nd ingredients first where the order is flipped and we first attend over ingredient embeddings e_L followed by image embeddings e_I .
- *Ingredient Decoder*. The ingredients can be represented as either a set (where the order of ingredients does not affect the outcome) or a list (where the order of ingredients does affect the outcome). Models that work either with a list of ingredients or with a set of ingredients is discussed belo:.
  1) *Ingredients List*
     A list of ingredients is a variable sized, ordered collection of unique meal constituents. A dictionary of ingredients of size N is defined as $D = \{d_i\}_i^N=0$. From which we can obtain a list of ingredients L by selecting K elements from $D : L = \{l_i\}_i^K=0$. Then L is encoded as binary matrix L of dimensions K × N with $L_i = 1$ if $d_j \epsilon D$ is selected and 0 otherwise. The training data consists of M image and ingredient list pairs $\{x^{(i)}, L^{(i)}\}_i^M=0$. The goal is to maximize the following objective:

$$argmax_{\theta I,\theta L} \sum_{i=0}^{M} logp(\hat{L}^{(i)} = L^{(i)} | x^{(i)}; \theta_I, \theta_L)$$

where θ_I and θ_L represent the learnable parameters of the image encoder and ingredient decoder, respectively. Since L denotes a list, $p(\hat{L}^{(i)} = L^{(i)}|x^{(i)})$, it can be categorized into conditionals. In theory, these conditionals are usually modeled with auto-regressive (recurrent) models. In this experiment as well the transformer model is chosen. A potential drawback of this formulation is that it is inherently penalized for order, which might not necessarily be relevant for ingredients.

2) *Ingredients Set*
A set of ingredients is a variable sized, unordered collection of unique meal constituents. A set of ingredients S is obtained by selecting K ingredients from the dictionary $D : S = \{s_i\}_{i}^{K}=_0$. S is represented as a binary vector s of dimension N, where $s_i = 1$ if $s_i \epsilon D$, and 0 otherwise. The training data consists of M image and ingredient list pairs $\{x^{(i)}, s^{(i)}\}_{i}^{M}=_0$. The goal is to maximize the following objective:

$$argmax_{\theta I, \theta L} \sum_{i=0}^{M} logp(\hat{s}^{(i)} = s^{(i)}|x^{(i)}; \theta_I, \theta_L)$$

The elements inside the set can be assumed to be independent of each other but this will not be the case all the time eg. salt and pepper frequently appear together.

B. *Training Method*

As seen earlier, ingredient set model will have dependencies among its elements. To account for the dependencies, ingredient list model can be used. But the drawback of this approach is that such a model is penalized for order. In order to remove the order in which ingredients are predicted, the outputs are aggregated across different time-steps by means of a max pooling operation(Softmax probabilities are pooled across time to avoid penalizing for order). To ensure that the ingredients in $\hat{L}^{(i)}$, are selected without repetition, the preactivation values are set to $-\infty$ for all previously selected ingredients before time-step k. This model is trained by minimizing the binary cross-entropy between the predicted ingredients (after pooling) and the ground truth. Including the eos(the last output from ingredient decoder) in the pooling operation would result in loosing the information of where the token appears. Therefore, in order to learn the stopping criteria of the ingredient prediction, an additional loss accounting for it is introduced called the eos loss. The eos loss is defined as the binary cross-entropy loss between the predicted eos probability at all time-steps and the ground truth (represented as a unit step function, whose value is 0 for the time-steps corresponding to ingredients and 1 otherwise). In addition to that, a cardinality l_1 penalty is incorporated, which is useful empirically. This model is referred to as set transformer (TF_{set}). The recipe transfomer is trained in two stages: In the first stage, the image encoder and ingredients decoder are pre-trained. Then, in the second stage, the ingredient encoder and instruction decoder are trained by minimizing the negative log-likelihood and adjusting θ_R and θ_E where θ_R and θ_E are learnable parameters.

C. *Dataset Description*

The training and evaluation of the models are performed on Recipe1M dataset. The Recipe1M dataset is consists of 1,029,720 recipes scraped from cooking websites. The dataset contains 720,639 training, 155,036 validation and 154,045 test recipes. Each of them containing a title, a list of ingredients, a list of cooking instructions and (optionally) an image. Out of which, only the recipes containing images are used, and recipes with less than 2 ingredients or 2 instructions are removed. Which results in 252,547 training, 54,255 validation and 54,506 test samples. The dataset was generated by scraping cooking websites, resulting in recipes that are highly unstructured and contain frequently redundant or very narrowly defined cooking ingredients (e.g. olive oil, virgin olive oil and spanish olive oil are separate ingredients). The ingredient vocabulary contains more than 400 different types of cheese, and more than 300 types of pepper. As a result, the original dataset contains 16,823 unique ingredients, which we preprocess to reduce its size and complexity. First, the ingredients which share first or last two words are merged (e.g. bacon cheddar cheese is merged into cheddar cheese); then, the ingredients that have same word in the first or in the last position are clustered (e.g. gorgonzola cheese or cheese blend are clustered together into the cheese category); finally the plurals are removed and ingredients that appear less than 10 times in the dataset are discarded. Altogether, the ingredient vocabulary is reduced from over 16k to 1,488 unique ingredients. For the cooking instructions, the raw texts are tokenized and the words that appear less than 10 times in the dataset are removed and replaced them with unknown word token. Special tokens are added for the start and the end of recipe as well as the end of instruction. This process results in a recipe vocabulary of 23,231 unique words.

D. *Performance Evaluation Methods*

In our report we are using Accuracy, F1 Score and IoU to evaluate our model. Accuracy is one of the simplest form of evaluation metrics. It tells that overall how often the model is making a correct prediction. F1 Score is the weighted average of Precision and Recall. This score takes both false positives and false negatives into account. The F1 Score lies between 0 and 1. Higher the F1 score, higher the precision, better the model. The Jaccard index, also known as the Jaccard similarity coefficient, is a statistic used for gauging the similarity and diversity of sample sets. The Jaccard coefficient measures similarity between finite sample sets, and is defined as the size of the intersection divided by the size of the union of the sample sets. Jaccard coefficient is also called as IoU (intersection over union). Higher the Jaccard % more the similarity. These evaluations are performed for the ingredients in the cooking instruction.

III. Implementation and Simulation

While attempting to recreate the results provided by the main paper [1], our approach deviated mostly from the one mentioned in the GitHub repository due to some hardware and software constraints.

To start off the dataset used to train the model is Recipe1M [2], which is a dataset containing about 1 million images, and hence nearly 180 GB in size. Since the size of the dataset was to big to be installed on our personal laptops, we decided to use ASU's Agave Cluster to carry out the training, validation and testing of the model.

The dataset is not quite readily available, so we were able to gain access to it after requesting for the dataset from one of the authors of the main paper [1]. These files were then downloaded and unzipped in Agave's scratch folder. We also cloned the GitHub repository [3] into the scratch folder in Agave. Next we installed all the software required that was mentioned in the requirement.txt file using the command given in the GitHub read.me . When using ASU's Agave Cluster, to execute anything on the compute node, we need to submit an SBATCH job. The training of the model required following these 4 main steps i.e.

1) Building the required vocabulary files of the ingredients and the recipes
2) Converting the image to LMDB
3) Training the model to predict ingredients from images
4) Training the generate recipes from images and ingredients

So to execute each of these processes, we needed to submit a new sbatch job to the Agave clusters. The exact code that was executed in the batch jobs will be given in the glossary. To build the vocabularies we used 2 GTX1080 GPUs with 40 GB memory and gave a time limit of 1 day for this job. In 12 hours the vocabularies were built. Next we again ran a batch job with same specifications for another 1 day to convert the images to LMDB. This job was completed in 4 hours. Finally we moved to training the model to predict ingredients from images. We initally used 2 GTX1080 GPUs with 40 GB memory and gave a time limit of 1 day for this job and ran the batch job. After 5 minutes of running we got an out of memory error. We simply assumed this was an issue with the amount of memory allocated so we increased the memory from 40 GB to 60 GB and ran the job again. This again resulted in failure due to an out of memory error.

At this point there were other computing nodes available in Agave that provided more memeory, and we simply deduced that we needed to first try using the one with great memory. We then ran the job using 2 Tesla K80 GPUs with 120 GB memory allocated. This again failed with the same error. When executing the python statement to train the model, we can specify the num_workers or the number of parallel threads that can run to execute the code. By default the code used 8 workers, but while going through the error log for the sbatch job, we noticed that all the workers were using around 40 GB of memory. This was strange for us because we were not able to train dude to insufficient memory is what the sbatch job claimed.

So after doing some more research we identified that some data structures take up more memory during run time as opposed to others. One such data structure was lists, we then decied to identify where all lists were being used in the training process, and wanted to see if we could make some code changes. The main train.py file was calling data_loader.py which was loading all the data from the pickled vocabularies and dataset. data_loader.py was mainly using lists, hence we changed all those to numpy arrays. We also found out that sometimes data deadlocks occur when multiple workers are used when using the pytorch dataloader library. This is not be confused with data_loader.py, which is a python file used to load the required data. To avoid the dead lock of data, we ran the training with 1 worker. The specifications of this job were the same as before but to be on the safe side we ran it for 3 days.

The job finally ran without the out of memory error, but it was immensely slow, it took 3 days to complete around 6 epochs of the first training. This was a training that was supposed to be running for 100 epochs, we then just ran the second training to generate recipes from images and ingredients while we brainstormed on how to approach this next.

At this point we had gone through all the possible avenues to decrease the amount of memory that was being used while training the model. We then started looking into the possibility of training the model but on a smaller dataset. The size of the dataset we decided should be small enough that it would run without any out of memory error, but it shouldn't be too small such that we receive no coherent results. We decided to scale down the dataset to a total of 7,215 images with 5061 images for training, 1065 images for testing, and 1089 images for validation. So to give a brief insight into how we arrived at this number, we went through the structure of the dataset. The dataset has a tree structure to the folders, where there are 15 main folders each containing 15 sub folders. This structure is followed till the essentially the root folders contain just 2 images. Due to this excellent structuring of the images, we then decided to use the images in the first two folders 0 and 1, thus reducing our dataset to nearly 10000 images.

Once we had decided to proceed with this idea, we now had the issue of creating the smaller dataset. This was not a simple issue of removing some folders from the data folder, because the data was read based on the pickled data set, not based on the number of images available. Hence we had to first figure out a way to extract only the required images from the dataset pickle files and create our own. We had to write new code in python which we called newfile.py and ran a new sbatch job called newfile.sh. Upon running this file, we then had to change all the lines of code that referred to or were working with the old pickle files, so that our new pickle file would be used instead.

Once we made the required changes in train.py and sample.py we now ran the model training again, with 2 Tesla K80

GPUs with 120 GB memory with a batch size of 75, number of epochs as 100 and num_worker as 1 for a time limit of 1 day. The training the model to predict ingredients from images took 6 hours to complete and only ran for 62 epochs, not a 100. This was because of a check placed in the training code to prevent the model from over fitting. We then ran another sbatch job with the same specification to train the model to generate recipes from images and ingredients. This training took around 5 hours and ran for 52 epochs. Finally we ran the sbatch job executing the file sample.py which gave us the F1, accuracy, jaccard and F1_ingredients score for our model.

But to obtain recipes generated from the model, the original code in github was given as a jupyter notebook. We cannot execute this in Agave, hence we had to then convert the code given in the jupyter notebook to a python file. Once we coverted this, we had to run it using another sbatch job. Also since this was not a jupyter notebook, the output file would not display the images for which the corresponding recipes were being generated. Hence we printed out the image file name, before we printing out the recipes. We then complied all the outputs we wanted into a format of image and corresponding recipe to add to the report. The model give around 4 recipes for a certain image and it also prior to providing the recipe tells us if the recipe is valid or not. It tells if its valid or not depending on the similarity of the image ingredients and the recipe steps.

IV. RESULTS

We first used the test set of images provided in the scale down version of the dataset to evaluate our model on the following metrics and obtained the corresponding values as shown in Table I:

Table I

Evaluation Metrics	Our Model	PreTrained Model
Accuracy	0.99404	0.8033
F1 Score	0.3510900	0.4861
F1 ingredients	0.006099	0.4908
IoU	0.212922510	0.3180

We observe that the accuracy obtained for the model is very high, this might imply that the model is slightly over-fitted. If we were to run this model again, we would less than 50 epochs around 20 to 30 epochs so as to avoid over-fitting. This accuracy is also higher than that of the pretrained model provided by Facebook research team. The F1 score of the model is pretty low compared to the F1 score of the pretrained model which had an F1 score of 0.4861. The jaccard metric is also called the IoU score. It measures the similarity coefficient score. The IoU score of our model is slightly less than that of the pretrained model which had a IoU score of 0.3180. The F1 score for the ingredients obtained for our model is extremely very low compared to that obatined for the pretrained model, which had a F1 score for ingredients as 0.4908. The model will first decode the ingredients from the images and train on those ingredients. As we used a smaller dataset, the number of ingredients that it was able to decode and train from greatly decreased, hence it was not able to accurately identify all the required ingredients.

To understand intuitively how well or how bad our model(trained on a smaller dataset) performed, we tested both the our model and the pretrained model on a set of images to see what kind of recipes they would produce. Through this exercise we were able to understand intuitively what influenced the training of the model, and if the results we obtained made sense. The images we used for this are the following

1) Enchiladas
2) Vada
3) Fish swimming in water
4) The Red Rocks

The recipes generated for the each of these images by our model trained on the smaller dataset and the pretrained model are shown in figures 3, 4, 5, and 6 for the images respectively. Our model in terms of ingredient prediction is pretty close to that of the pretrained model. If we observed the results obtained of the red rocks which the pretrained model believes to be peanut butte cookies, the ingredients generated by our model is very close to the ingredients predicted by the pretrained model. As similar theme is observed in the results comparison of all these images. We also felt that if we had trained our model on a more diverse dataset of images, encompasing dishes from all portions of the world, it would in turn be able to predict the ingredients more accurately for dishes like Vada, which is a traditional Indian dish.

Fig. 3. Chicken Enchiladas

V. DISCUSSION AND CONCLUSION

We expected our results to perform poorly compared to Facebook's Recipe1M-trained generative model. However, while our model did not perform as well, we can see how accurate it was in generating ingredients for each recipe considering the much smaller dataset it was trained with. In fact, even the recipe generated by our model, although very rough and unspecific, reflected our model's ability to recognize how these ingredients should combine together.

Fig. 4. Aquarium

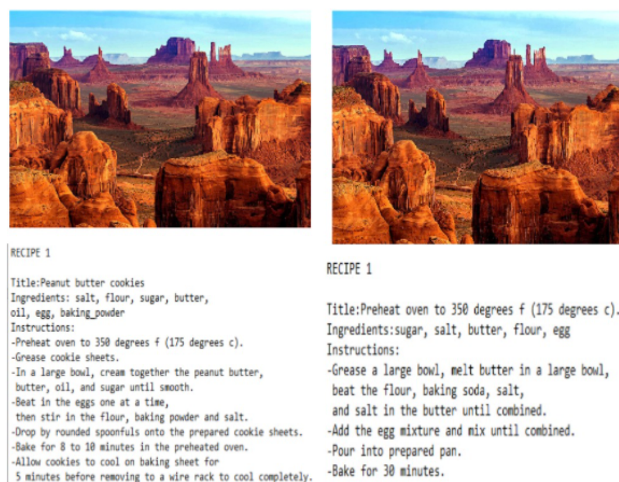

Fig. 5. Grand Canyon

What's also interesting to note is that our model did not attempt to title dishes with specific names but rather with vague descriptions. We expected the dish naming to be similar to the Facebook's model's the model attempts to name the dish presented in the image. This highlights the difference between a retrieval-based model versus a generative-based model. If our model was a retrieval-based model, then it would have tried to pick a recipe from its limited dataset that has the highest correlation with its given image and output that recipe no matter how low that correlation is. Instead, our model decided that it did not have enough information to confidently name the dish in the given image a specific name hence the vague description. The reason why Facebook's model outputted a

Fig. 6. Sambar Vada

specific name for each dish was because it had access to a larger pool of information (e.g the entirety of Recipe1M).

Highlighting the difference between generative models and retrieval models may be the key takeaway from this project. In order for the field of the computer vision to progress, we cannot allow the model to be limited by its given dataset. Future models tackling similar problems to Inverse Cooking must be able to create their own inferences instead of regurgitating results from a dataset.

Future work that we can do for this project is run two different transfer learning models. We would try to train the model on MobileNet since it is targeted towards edge devices like embedded systems and mobile phones. We would also try to train the model on ResNext as it aims to go wide rather than deep. Better recognition may provide better recipes. We also would like to train all of these models, both new and old, on Recipe1M+ to see if training on a larger dataset provide better recipe generation.

References

[1] A. Salvador, M. Drozdzal, X. Giró-i-Nieto, and A. Romero, "Inverse cooking: Recipe generation from food images," CoRR, vol. abs/1812.06164, 2018, [Online]. Available: http://arxiv.org/abs/1812.06164.

[2] Marín, Javier Biswas, Aritro Ofli, Ferda Hynes, Nicholas Salvador, Amaia Aytar, Yusuf Weber, Ingmar Torralba, Antonio. (2018). Recipe1M: A Dataset for Learning Cross-Modal Embeddings for Cooking Recipes and Food Images.

[3] Code: https://github.com/facebookresearch/inversecooking

[4] Repository overview and quickrun guide: https://github.com/facebookresearch/inversecooking/blob /master/README.md

[5] Layers: http://wednesday.csail.mit.edu/temporal/release/recipe1M_layers.tar.gz

[6] Ingredient Detection: http://wednesday.csail.mit.edu/temporal/release/det_ingrs.json

[7] Training data link: http://wednesday.csail.mit.edu/temporal/release /recipe1M_images_train.tar

[8] Testing data link: http://wednesday.csail.mit.edu/temporal/release /recipe1M_images_test.tar

[9] Validation data link: http://wednesday.csail.mit.edu/temporal/release /recipe1M_images_val.tar

VI. Disclaimer

The contents of this report have been equally shared among the individuals in the group.

Link for Presentation:
https://docs.google.com/presentation/d/1sIv79rMpwgpI1TARtS1sWsB5YQjkTP3sLr9ndYiLzHs/edit?usp=sharing

ReadMe and Link to Github Code:

Team 5: Inverse Cooking

This README file will guide you to install the necessary dependencies to run the software, and recreate the same results that we did. We will be using ASU's Agave Cluster to train this model.

Installing Python 3.6 Dependencies

We are using Python 3.6. To install the correct dependencies, run the follow the command:

`python3 -m pip install --user -r requirements.txt`

Pulling the Inversecooking Repository from GitHub

MAKE SURE TO DOWNLOAD EVERYTHING INTO SCRATCH SPACE

Run this command to pull the Inversecooking repository from GitHub:

`git clone https://github.com/facebookresearch/inversecooking.git `

If you would like to test out their pretrained model, make sure to follow their directions, download the correct files (e.g `modelbest.ckpt`, `ingr_vocab.pkl`, `instr_vocab.pkl`), and place them in the correct directory. We will provide more directions on how to the pretrained model later in the README file.

Downloading the Recipe1M Dataset

There are quite a few of `.tar` and `.json` files to download before getting started. The commands to download them will be listed below:

`wget http://wednesday.csail.mit.edu/temporal/release/recipe1M_layers.tar.gz`

`wget http://wednesday.csail.mit.edu/temporal/release/det_ingrs.json`

`wget http://wednesday.csail.mit.edu/temporal/release/recipe1M_images_train.tar`

`wget http://wednesday.csail.mit.edu/temporal/release/recipe1M_images_test.tar`

`wget http://wednesday.csail.mit.edu/temporal/release/recipe1M_images_val.tar`

These files should extracted using either command `tar xvf <file>.tar` or command `tar xvzf <file>.tar.gz` and the extracted files should be placed in the `inversecooking/data ` folder. The directory hierarchy should look like this:

```
\---data
        +---det_ingrs.json
                layers1.json
                layers2.json
                images
                +---train
                        +---<training data>
                +---val
                        +---<validation data>
                +---test
                        +---<testing data>
```

Running SBATCH Scripts

To take advantage of Agave's resources, we need to be able to submit jobs through SLURM, a workload manager. We have three SBATCH scripts that needs to be run in this order: `build.sh`, `newdata.sh`, and `trainmodel.sh`. These scripts will be used to train the model with the **reduced** dataset and obtain some benchmarks for that model. Because our model has two parts, two separate trainings will occur and can be seen in the `trainmodel.sh` script. Before running these scripts, however, make sure to change the directories in the arguments to the directories in your directory. To run an SBATCH script, run the command `sbatch <file>.sh`. If there seems to be a problem in allocating a job to a node, you can tweak the parameters before the `python3` commands to find the best available node. To know which nodes are available, please use this link: https://rcstatus.asu.edu/agave/smallstatus.php.

To run the pretrained model, do **not** use demo.ipynb as errors tend to pop up that way. Instead, run the SBATCH script `pretrained.sh`.

How your output should look like

The output and error logs for these jobs will be located in `inversecooking/checkpoints/<model name>/logs`. The training will output to files `train.log` and `train.err`. The benchmarking Python script will output to files `eval.log` and `eval.err`. You can compare your outputs to our files in the directory `logs` in our zip file.

Link to Github Code:

https://github.com/ArkanDH/Team5-Inverse-Cooking-Stuff