What’s in Your Fridge?

James Sullivan, Zachary Chilton, Young Joon Lee, Royce Hong, Erick Li, Zander Tedjo

University of Texas at Austin, EE460J Data Science Lab, Dr. Alexandros G. Dimakis

Abstract

TODO

Hello, and welcome to our final project video for “What’s in your Fridge?”, which seeks to answer the question of what we can put together from what we have.

Problem Description

The idea is simple; a user wants to be able to take a picture of the inside of their fridge to figure out what recipes they can make from its contents. Our job is obviously the difficult part; figuring out the areas of their image that could be a grocery item, figuring out what that grocery item is, and then taking all of those identified items to search for what they can make.

Overview

Our project can be broken down into three parts. In order to detect food items within a large image of a fridge, we used the RCNN model and its variations for region proposal. (RCNN stands for regional convolutional neural network). Next, we used Tensorflow and a pretrained model to classify images of food items. Finally, we used the Spoonacular API to generate recipe ideas from a list of ingredients. The primary reason why we decide to split our process into three steps rather than simply use a RCNN to detect and classify food is because of datasets, or lack thereof. While there are many food datasets available online, there weren’t many that were properly formatted with bounding boxes and labels to reliably have one model do everything. As a result, we decide to use two separate models to perform object detection and classification.

Datasets

Thankfully, food is pretty common in our lives, so there were already quite a few datasets dedicated to classifying it. Certain datasets were used to train the region proposal network and others were used to train the food classification. We explored many sources of data, and will highlight the most significant ones.

Region Proposal Datasets

For the region proposal part, there weren’t as many datasets that had positional information, so we were only able to work with the COCO and MVTec datasets. The COCO dataset is a large-scale object detection, segmentation, and captioning dataset. COCO stands for Common Objects in Context. It contains 1.5 million object instances and 80 object categories. COCO and MVTec were annotated using a .json file, so for the faster rcnn model we had to reformat the dataset into .xml files.



*Figure 1 - COCO Dataset*

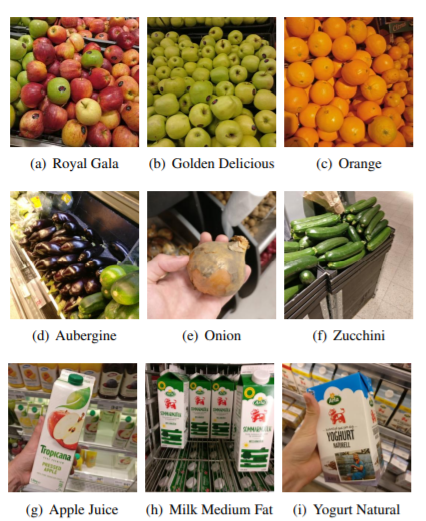
The MVTec Densely Segmented Supermarket (D2S) dataset is a benchmark for instance-aware semantic segmentation in an industrial domain. It contains 21,000 high-resolution images with pixel-wise labels of all object instances. The objects comprise groceries and everyday products from 60 categories. The benchmark is designed such that it resembles the real-world setting of an automatic checkout, inventory, or warehouse system. An example image is shown below in Figure 2.



*Figure 2 - MVTec Dataset*

Classifier Datasets

For our classifier dataset, we chose to go with Marcus Klasson’s Grocery Store Dataset. This dataset contained 5125 natural images from 81 different classes of fruits, vegetables, and carton items that you might find in your fridge. These 81 classes are then further divided into 42 coarse-grained classes, where rather than dividing by specifics such as “Gala Apples” or “Granny Smith Apples”, both would simply be under “Apples”. This dataset worked well for us because their natural presentation in the images was similar to how they would be presented in a fridge, making our classification more effective.

****

*Figure 3 - Klasson’s Grocery Store Dataset*

At first, we removed the classes that weren’t fruits or vegetables for the sake of being able to fit our dataset on GitHub. Later, we were able to use all of the classes and import them into our Google Colab notebook. We also flattened the fine-grained versus coarse-grained folder structure so that all data was immediately accessible one level into their directory. This was important so that the model could automatically identify the images and their labels, with the folder name serving as the class name.

The other datasets that we considered using for training our food classifier included the Food 101 dataset and the Freiburg Groceries dataset. The Food 101 dataset has 101,000 images in total. It's a food dataset with 101 categories. Each type of food has 750 training samples and 250 test samples. It was used in the paper "Food-101 – Mining Discriminative Components with Random Forests" by Lukas Bossard, Matthieu Guillaumin and Luc Van Gool. One challenge we had with this dataset was that many of its images were of completed dishes, not raw ingredients.



*Figure 4 - Food 101 Dataset*

The Freiburg Groceries dataset was part of a paper written by Philipp Jund, Nichola Abdo, Andreas Eitel, and Wolfram Burgard from the University of Freiburg in Germany. The dataset consists of 5,000 images covering 25 different classes of groceries, with at least 97 images per class. Some classes include “Beans”, “Cake”, “Pasta”, and “Spices”. The images were from real-world settings and include a large variety of perspectives, lighting conditions, and degrees of clutter. This dataset shows a lot of potential, but a challenge we had was its small number of classes and the broadness of those classes.



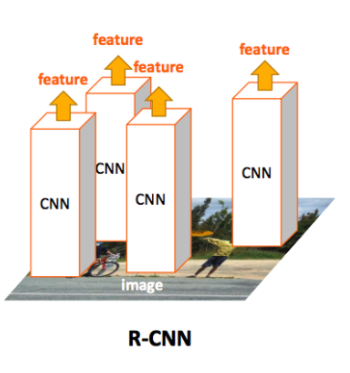
*Figure 5 - Freiburg Groceries Dataset*

Region Proposal Models

We explored several region proposal models in order to detect objects of interest within an image. Some models worked better than others, and we explain the differences below. Ultimately, we decided to go with the Faster R-CNN model.

R-CNN

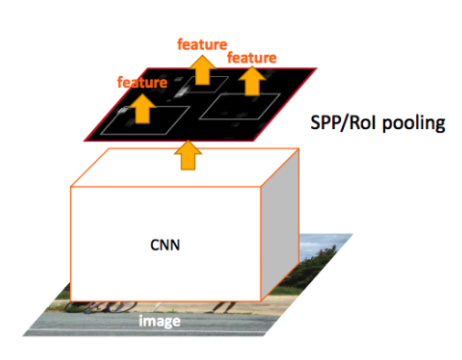
The idea of R-CNN is to extract region proposals, usually about 2000, warp those into a square, and then feed those into a convolutional neural network. This takes a lot of time to train, as each image in your training set would require 2000 classifications. Additionally, with each proposed region being a fixed part of the algorithm, no learning is done as for where to look next. Figure 6 shows this process, where fixed regions are fed into a convolutional neural network repeatedly.



*Figure 6 - R-CNN Model*

Fast R-CNN

Fast R-CNN improves upon R-CNN by feeding the entire input image at once to the convolutional neural network and then generating a feature map. This feature map then identifies the proposal regions, warps them into squares, and then attempts classification. The ability to prevent 2000 classifications through the use of a convolution neural network saves a lot of time, with some results showing a speedup factor of nearly 10x. In Figure 7, we see how this model is different from the previous one. Now, the entire image is fed at once in order to identify regions and attempt classification on those.



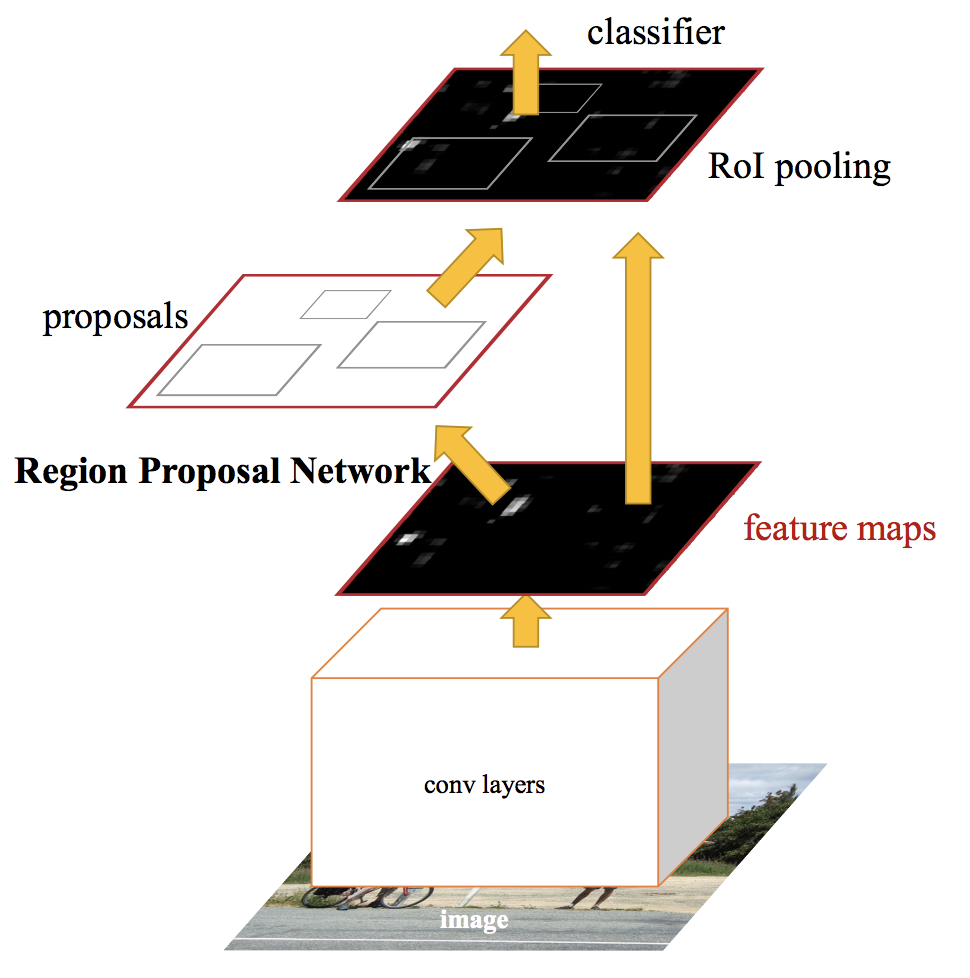
*Figure 7 - Fast R-CNN*

Faster R-CNN

Faster R-CNN improves upon both of the previously mentioned methods through the elimination of the use of selective search, which is a slow and time-consuming process. An object detection algorithm is instead used to allow the network to learn the region proposals.

Similar to Fast R-CNN, the image is used to generate a convolutional feature map, but instead of using selective search, an entirely separate network is used to predict region proposals. This is akin to the difference between grid search versus Bayesian Optimization in order to find hyperparameters. This is the model we decided to use for object detection.

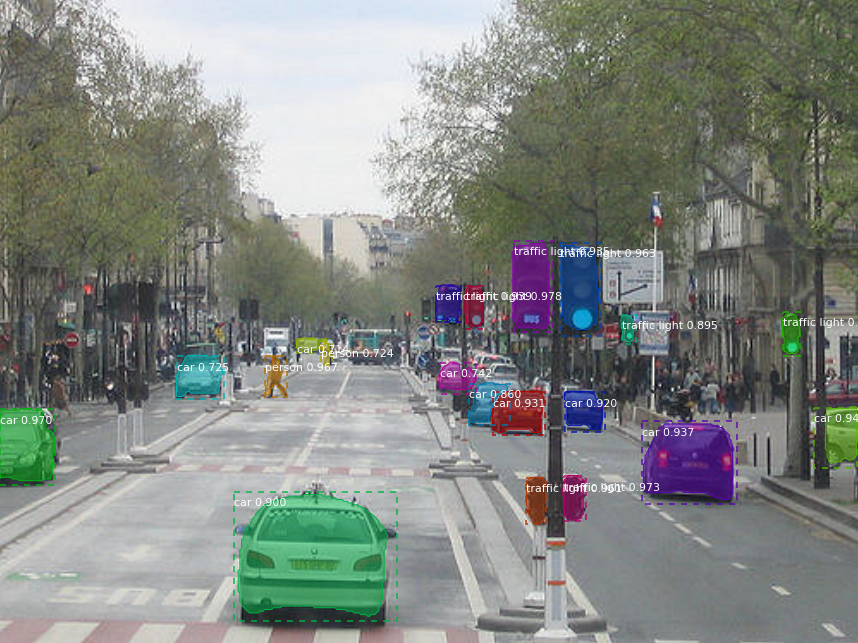
The model we decided to use for object detection was an FRCNN model implemented using Keras. This model was convenient for our project because the RPN could be trained separately from the rest of the network. [Continue]



*Figure 8 - Faster R-CNN Model*

Mask R-CNN

Mask R-CNN is an extension of Faster RCNN in which an extra branch is added to predict the outline of the object on top of its bounding box. The implementation we tried was built using TensorFlow and pretrained on MS COCO. We then tried to train on top of MS COCO the MVTec dataset. A major drawback with using MRCNN was it could only be trained using datasets with images that had fully classified and labeled bounding boxes, which made the scope of usable datasets extremely limited. Training this model turned out to be extremely slow and with too few computational resources to operate this model efficiently, so we decided to not to use this model.



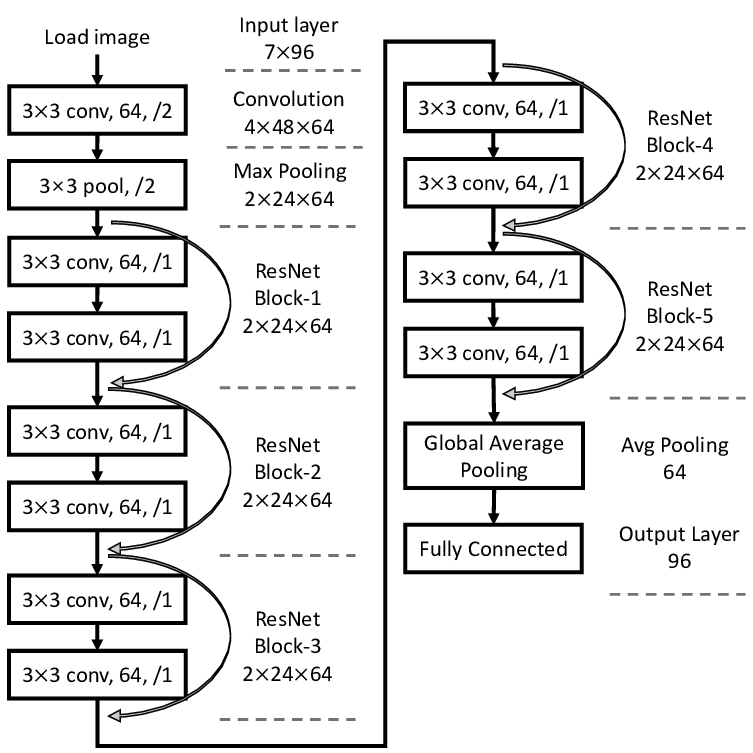
*Figure 9 - Mask R-CNN Example*

Classification Models

For classifying images of foods, we chose to use transfer learning. This involves taking a pretrained model and modifying to our specific application. Transfer learning is an effective method because it greatly reduces the time and cost of training a large and powerful network. We tried two approaches for image classification.

ResNet

First, we used transfer learning from a pretrained ResNet model with Pytorch. ResNet is a residual neural network that uses skip connections to jump over layers. Typical ResNet models are implemented with double- or triple- layer skips that contain nonlinearities such as ReLU and batch normalization in between. One motivation for skipping over layers is to avoid the problem of vanishing gradients, which greatly hinders the process of backpropagation. ResNet is able to reduce this problem by reusing activations from a previous layer until the adjacent layer learns its weights. Another benefit of ResNet is that skip connections effectively simplify the network. This speeds learning by reducing the impact of vanishing gradients, as there are fewer layers to propagate through. The network then gradually restores the skipped layers as it learns the feature space.

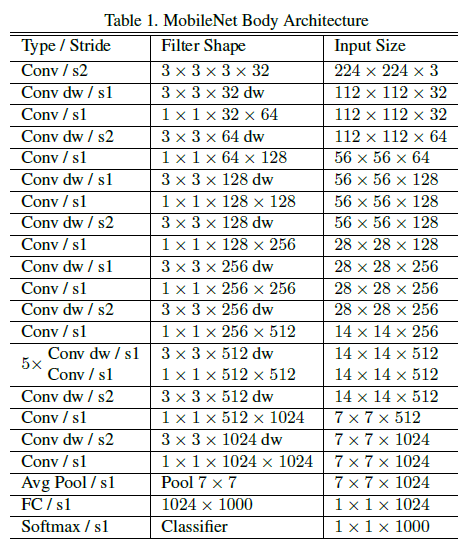


*Figure 10 - ResNet Architecture*

We figured ResNet would be a good option to explore since it was previously trained on ImageNet, which is a large well-known dataset that contains images of some food classes. In order to adapt the pretrained model to our problem, we modified the last fully connected layer of the ResNet architecture to match the number of food classes that we had. We then retrained this new network over a number of epochs and were able to achieve an accuracy of 0.68. As a result, the model performed decently well, but had room for improvement.

MobileNet

Our second approach used a pretrained MobileNet v2 model trained on ImageNet. We found this model on Tensorflow Hub. We took all the images from the grocery dataset and retrained the MobileNet v2 model with these images. Before feeding these images into the model, we resized all the images so that they were suitable for the model and performed some data augmentation (random distortions of an image each time it is read) to help improve the performance during training. With this model, we were able to achieve an accuracy of 0.92 on the validation set. This was a significant improvement over the Resnet model we tried, so we saved the final MobileNet v2 model and decided to use it as our image classifier.



*Figure 11 - MobileNet Architecture*

Recipe API

The API that we used is Spoonacular, which uses ingredients and other parameters to search for recipes. This is the last step in our pipeline, following region proposal and classification. We used their findByIngredients endpoint, which included an option to either maximize used ingredients or minimize missing ingredients. We went with the latter due to the fact that our project seeks to create recipes out of what is present in your fridge. This API allowed us to nicely wrap up the output of our project by giving us a simple way to search for recipes from our classified ingredients.



*Figure 12 - Spoonacular API*

Results/Demo

TODO

Challenges/Next Steps

One issue that we ran into was moving images from our region proposal model to our classifier model, as our region proposal network would provide us with “region of interest” coordinates, which weren’t an exact 1-to-1 mapping to the original image like we were hoping.

Another improvement we could make is gathering more datasets for image classification. A larger and more diverse set of data would enable us to increase the number of food classes as well as further increase our accuracy, thus creating a more powerful classification model.

References

* <https://arxiv.org/abs/1901.00711>
* <http://aisdatasets.informatik.uni-freiburg.de/freiburg_groceries_dataset/>
* <https://www.kaggle.com/dansbecker/food-101#food-101.zip>
* <https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e>
* <https://www.mvtec.com/company/research/datasets/mvtec-d2s/>
* <http://cocodataset.org/#home>
* <https://spoonacular.com/food-api>
* <https://towardsdatascience.com/overcome-overfitting-during-instance-segmentation-with-mask-rcnn-32db91f400bc>
* <http://kaiminghe.com/iccv15tutorial/iccv2015_tutorial_convolutional_feature_maps_kaiminghe.pdf>
* <https://towardsdatascience.com/review-mobilenetv1-depthwise-separable-convolution-light-weight-model-a382df364b69>