Project Title: Pantry Recipes

Nicholas Mataczynski, Connor Boucher

Problem Selection

Our project started with the idea of being able to detect multiple food objects from the same given image. The goal was to create the base for a program that, when given an image of a user's pantry, could take an image, locate potential ingredients, and then provide potential recipe ideas that could be made with the available ingredients. We chose this project idea as we were intrigued by the homework assignments revolving around image classification and wanted to expand on the concept; logically, the next step was to find multiple objects from one image rather than just one.

Dataset

Initially, we wanted to use our own dataset in order to create a custom model. We started with the Fruit Recognition dataset on Kaggle, giving us thousands of pictures across fifteen different fruit classes to use. We needed to edit this dataset slightly, as certain fruit classes had multiple folders of images rather than just one, which caused issues when attempting to find and load images from file paths. The changes made were to delete all folders except one from each class so that data was stored uniformly.

Due to time constraints from the time spent on the scrapped initial model, we were unable to make a custom model for all the food classes we had wanted. The choice we had to make was looking into pre-trained models that more closely represented the object classes we wanted to detect. The issue with this is that they are far less flexible due to us being unable to add additional classes, but they have the positive of being far more accurate. We started with the COCO dataset, a large scale dataset useful for segmentation and object detection across a wide array of object types such as vehicles, animals, and most important for our work, food. Eventually, we changed this to using the weights from GrocerEye, a YOLO model that works with raw

ingredients and was closer to what we had in mind for making recipes from pantry ingredients.

Problem Solving Techniques/Models

Our initial thought was to create a model the same way we had previously in class to identify food items, but after doing so we attempted to integrate it with YOLO in order to segment the image and found that the YOLO model would not fit to the training dataset we had set up regardless of what we tried. If we had researched more early on, we would have seen that we needed to specifically train the model for use with YOLO by using a dataset that had bounding boxes already set up. We stopped working on this initial idea, removed the YOLO model and cleaned up the file to make it more readable, and have saved the file to GitHub as “Image_Classification_Scrapped”.

In order to create a custom weights file, the creator must install a program such as Microsoft’s Visual Object Tagging Tool (VoTT) and manually draw boundaries around what the computer should identify in each image. An example can be found on How to Train your Own YOLOv3 detector from scratch where the writer performs this process to create an object detector for cats. In order to get good results, we would need to manually outline at least one hundred images per class we want to detect. Because we could not find a readily available dataset that had this bounding box data, and because of time constraints, it was not feasible for us to train our own model using this method.

At this stage we also looked into other potential methods for segmentation such as SDD but found that we could not use them for similar reasons to YOLO, so we decided to stick with YOLO. In order to be able to progress with the project we initially used the pre-trained YOLO model trained on the COCO dataset. This worked but was limited due to a small selection. The best option we found to remedy this was moving over to a new pre-trained model from a project called GrocerEye, which had a larger array of food classes. It still was not ideal, but it was enough that we could move on with the rest of the project.

After we had the pre-trained model set up, the next step of the program consisted of parsing and visualizing the results. We load in the trained YOLO model and use it to perform segmentation of the image into a collection of images called blobs. We also call getLayerNames() on the model, which gives us names of each layer and can be used by calling forward(layer_names) on the model to perform a forward pass of the YOLO model's object detection. This method returns to us a list of output layers, which we then loop through to get each layer and then loop through again to get each detected object in each layer. We can pull out the class that was detected and the confidence in the detection from these objects, which we both store to respective lists if the confidence value is high enough. Each detection also contains the center coordinates as well as width and height of each boundary box, which we extract, convert to be relative to the image's size by multiplying by the width for X and height for Y, and also append to a list to keep track of all boxes. These boxes are run through the OpenCV function cv2.dnn.NMSBoxes(), which checks for boxes with very high degrees of overlap, which we chose to use a threshold of 0.2, and removes the weaker of the two boxes. The final boxes are flattened and converted into rectangle objects, which we then display using the OpenCV function cv2.rectangle() along with the confidence and class name using cv2.putText().

The last thing we did was set up a simple GUI within the Jupyter Notebook to be able to take an image, predict on it using the model, and recommend recipes that the user could make based on what was found. This approach is limited by the small number of available foods which cannot on their own fill out many recipes. Because of this limitation the current recipe database we have was created by us and only contains a few items that fit within our restrictions. Currently the GUI is more of a proof of concept, but would be able solve our problem in the future if the model had a greater range and we were to implement a more robust recipe database.

Solution/Results

The initial model we created and scrapped was able to achieve an accuracy of approximately 98.11% after running three epochs using 35525 of the 44406 images in the dataset for training. The model was compiled using the Adam optimizer and uses a Conv2D layer with the ReLU activation function and a MaxPooling2D layer to help prevent overfitting. We use a fully connected dense layer at the end with the ReLU activation function.

The pre-trained model we acquired works well for our purposes aside from its limited scope. It generally has good accuracy when identifying food, and performs well even if there are many food items in a single image. That being said, it is not perfect. It often messes up classification on items that are somewhat obscured, and has trouble with items that resemble one another such as sugar and flour. Despite these limitations it still performs much better than the COCO model for our purposes even if it has lower consistency because this pre-trained model has a much better range of 25 classes compared to the default COCO model, which only has 8. In addition, most of the classes in this model are common ingredients, whereas COCO only has a small number of classes that could be ingredients.

The end result of our work is a program that is able to recommend a limited number of recipes from the limited number of food classifications that came with the pre-trained model. These limitations are the result of us being unable to produce our own model with the foods that we wanted. As such our program is not really useful in its current state, but it does serve as a proof of concept. The more food classifications that the model has, the more useful the program would become.