Image2Recipe

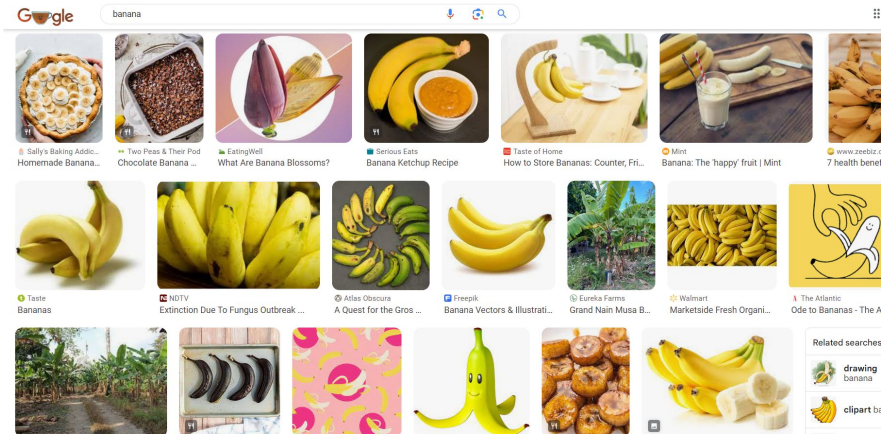
Dennis Phun, Christine Widden, Yohan Sofian, Gordon Luu

Original Goal

- The app will identify ingredients from user pictures and suggest recipes accordingly
  - Users upload images of ingredients they have
  - App analyzes these images to identify the ingredients present
  - Based on the identified ingredients, the app recommend a list of recipes to the user (using an API and possibly some generation)
- Ultimate goal would be for users to upload a single picture (e.g. a refrigerator or tabletop containing multiple ingredients) and have recipes generated
  - Object detection typically requires annotating images with bounding boxes and labels
  - Different scaling with a sliding window may work

Dataset

- Problem: No large, public dataset for food ingredients
- Solution: Compiled our own via crawling images from search engines (mainly Google and Bing)



- Sourcing ourselves isn't as simple as just web scraping
  - Drawings, fully cooked meals, cluttered images, and otherwise poor data
  - Scraped images are often clear, brightly-lit, and place the foods on white backgrounds unlike how the food may appear when photographed on someone's kitchen counter

Processing the Dataset

- Initial quick, manual scan of the images to remove any that stood out
- Used our model to help filter further
  - Couldn't simply remove every image, or it would bias the model towards certain things, such as the common forward-facing, single-object images
- We didn't want to annotate the images with bounding boxes, so that was the main thing we kept in mind while creating our models

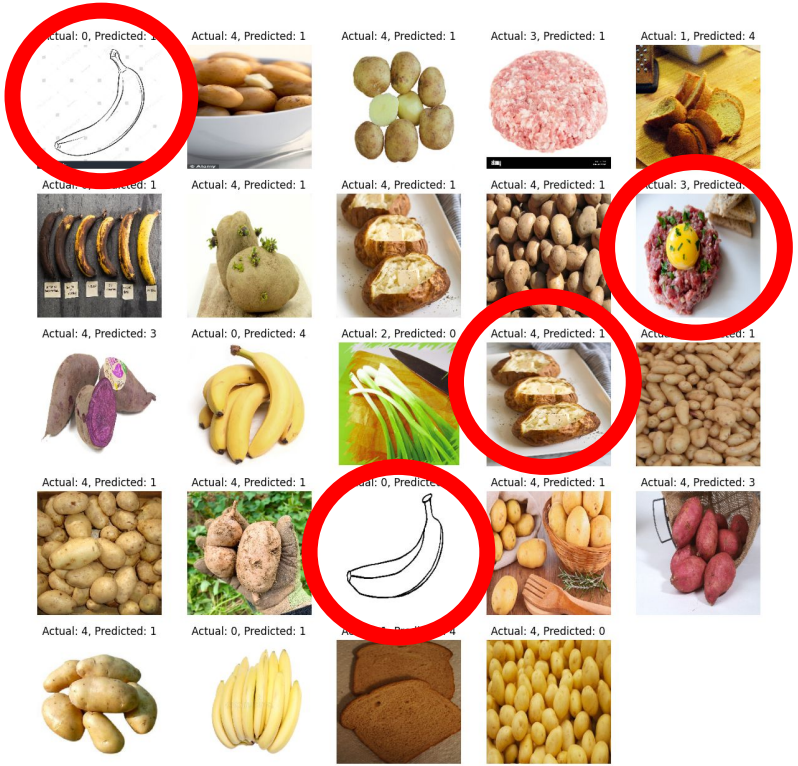
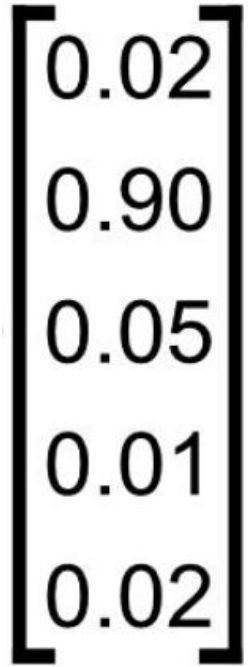


Image Classification with a CNN

Model: "sequential_2"

Layer (type)	Output Shape	Param #
conv2d_8 (Conv2D)	(None, 222, 222, 32)	896
conv2d_9 (Conv2D)	(None, 220, 220, 32)	9248
max_pooling2d_4 (MaxPooling2D)	(None, 110, 110, 32)	0
dropout_6 (Dropout)	(None, 110, 110, 32)	0
conv2d_10 (Conv2D)	(None, 108, 108, 64)	18496
conv2d_11 (Conv2D)	(None, 106, 106, 64)	36928
max_pooling2d_5 (MaxPooling2D)	(None, 53, 53, 64)	0
dropout_7 (Dropout)	(None, 53, 53, 64)	0
flatten_2 (Flatten)	(None, 179776)	0
dense_6 (Dense)	(None, 64)	11505728
dense_7 (Dense)	(None, 128)	8320
dropout_8 (Dropout)	(None, 128)	0
dense_8 (Dense)	(None, 1275)	164475

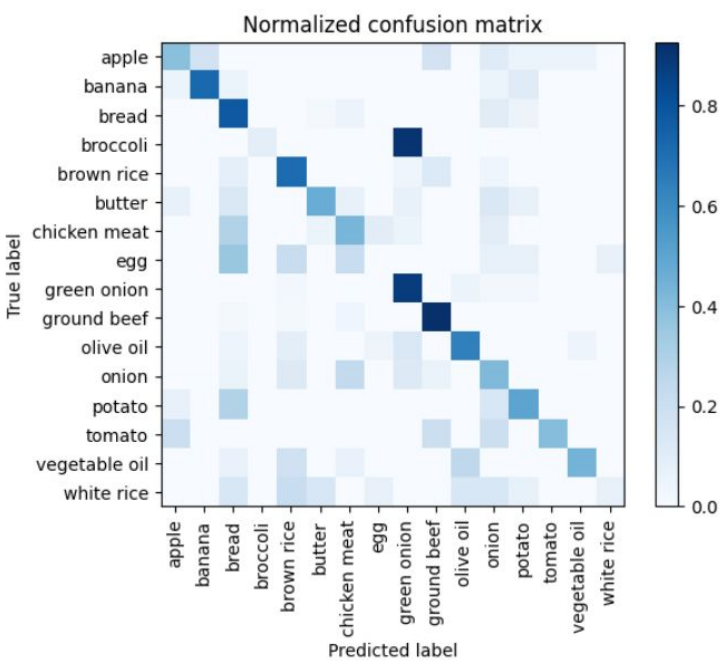


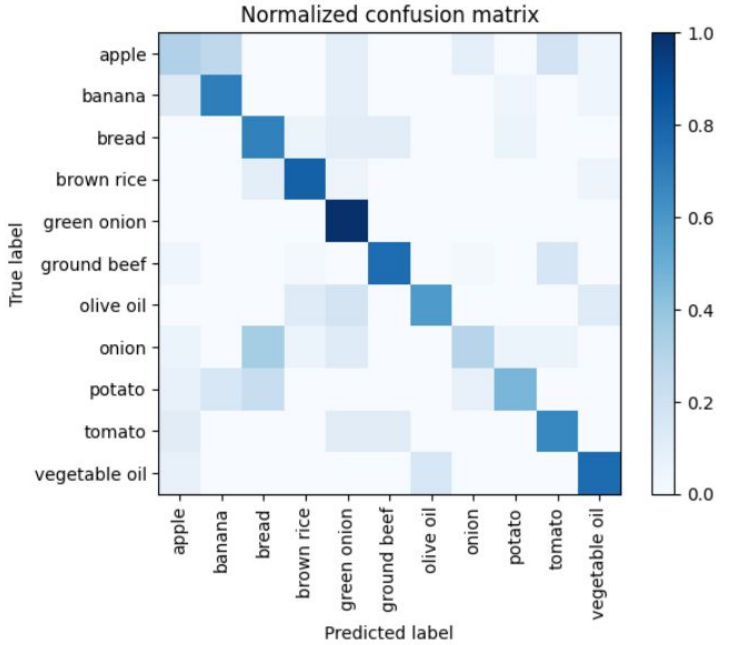
green onion

Output layer uses a softmax activation function

Image Classifier Accuracy

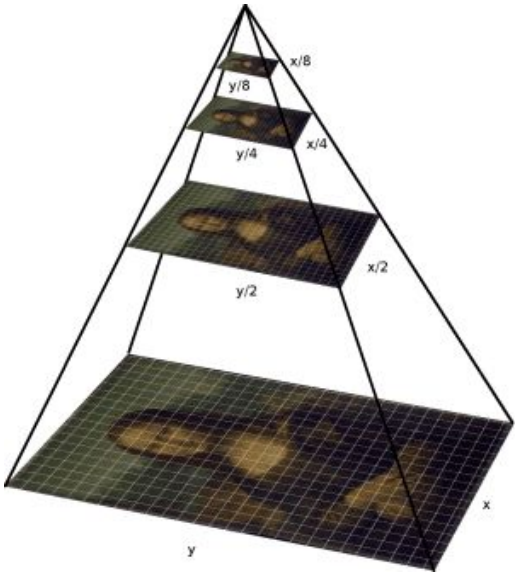
90% with 5 classes → 62% with 16 classes → 76% with 12 classes





Preprocessing Inputs

Image Pyramid



Sliding Window

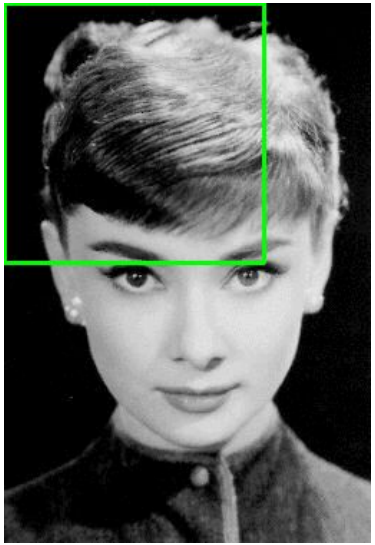


Image Classifier

CNN

Limitations

- Slow, not a very useful implementation for our use-case
- Aspect ratios must be maintained among the levels of the image pyramid
- Window size and stride must be small enough to detect objects, but large enough to run at a decent speed
- Performs poorly when attempting to detect small or overlapping objects
- Built on top of CNN

Future Improvement

- Path #1:
  - Refine the CNN by cleaning up and adding to the dataset (more classes, images per class, different orientations, etc.)
  - Tune the sliding window
  - Adding a border to the images so the sliding window works on edges
  - Combine the levels of the image pyramid (non-maximum suppression, a technique that keeps only the most probable bounding box of overlapping ones)
  - Custom confidence scores for the bounding boxes
- Path #2:
  - Training a model that incorporates a pretrained, SOTA object detection model
    - Requires annotating each training image of a custom dataset
      - May be easier than finding hundreds of new images
    - Better performance in terms of speed and accuracy

Questions?