G-75
April 11, 2024

Nutrition Label Recognition App

Authors:
Paul Airuehia
Matthew Arabo
Nicholas Veselskiy

Abstract:
This project is an application that processes images that contain nutrition labels and returns the information in a digital format. The application is user-friendly, taking the user's image as input and processing it in a server. Rotation angle detection is used on the image, the nutrition label is then detected and cropped using the YOLOv5 AI model. Following the detection, some filters are applied so that it can be in an optimal format for the Tesseract OCR engine, which extracts text from the image. The text is then parsed and necessary edits are applied before being sent back to the user. Through testing with various types of images, the findings show that accuracy in returning proper nutrition information decreased as lighting and rotation in images got worse. It was found that the dropoff in accuracy due to rotation begins to stagnate once the rotation reaches a certain point.

Introduction:

The purpose of this application was to provide a simple way for someone to store nutritional information. Since inputting all the information manually can be time-consuming, taking a picture and having the data produced instantly was an optimal solution. There also did not seem to be many relevant applications specific to nutrition labels, but rather taking pictures of barcodes and requiring connection to pre-existing databases, which is not what the project is focused on. By providing the option to take photos of the labels and store them on the user's device, the application provides a baseline for implementing future nutrition label-related features of any kind. Many challenges arose when dealing with label detection and formatting. Training the model enough so that it could detect the label itself required an abundance of dataset images. Being able to identify all the relevant text and disregarding the non-relevant text after it had been extracted from the OCR engine also proved to be difficult.

Background:

Nutritional information for an individual's consumption can be important for a variety of reasons. The paper focuses on nutritional status and explores the importance of its evaluation in diagnosing nutritional-related medical issues [4]. An application that can streamline the process of recording the nutritional information from the label can be useful in these types of evaluations.

Architecture:

The project utilizes a client-server architecture with two main components:

- React Native App (NutritionApp): Developed using React Native and Expo for cross-platform mobile functionality. It allows users to capture or upload photos of food items and displays extracted nutritional information.
- Python Flask REST Server: Acts as the backend processing unit. It handles communication with the app, performs label identification, image processing, and extracts nutritional data.

Computer Vision Components:

- Image Rotation (By text alignment): The server employs computer vision algorithms to rotate the uploaded image, aligning the nutrition label for improved detection.
- Object Detection (YOLOv5): Custom-trained YOLOv5 model identifies the bounding box of the nutrition label within the image using object detection.

- Text Extraction (PyTesseract/Google Tesseract OCR): Utilizes PyTesseract and Google Tesseract OCR engine for extracting text from the cropped nutrition label.

Data Flow

1. User Interaction: The user using the NutritionApp either takes a picture or uploads a photo containing a food item with a nutrition label.
2. Image Upload: The app sends the image data to the Flask server.
3. Image Processing: The server first verifies successful image reception. It then performs image rotation using computer vision to align the nutrition label for better detection. YOLOv5, the detection model, then identifies the bounding box of the nutrition label (if present) or determines if no valid label is found.
4. Label Extraction: If a label is identified, The server crops the image containing the nutrition label. Further computer vision techniques with the PyTesseract OCR engine then extracts all text within the cropped image. If no label is identified, the server prepares a response informing the app.
5. Data Parsing: The extracted text undergoes filtering to identify relevant nutritional information. This information is then parsed and structured into a JSON format.
6. Response and Display: First the server sends a response to the app: If a label is found, the response contains the JSON object holding the extracted nutritional details. If no label is found, the response indicates the unsuccessful detection. The NutritionApp interprets the server response. If successful, it displays the extracted nutritional data to the user. If unsuccessful, it informs the user about the missing label. Additionally, the app can optionally store the received nutritional data persistently for future reference.

This architecture allows for a modular and scalable solution where the mobile app focuses on user interaction and presentation. At the same time, the server handles the image processing and data extraction.

Methodology:

For building the AI model we used YOLOv5 object detection. We needed to train the model on images of nutritional labels that had the label annotated in the Yolo format. We initially found a public dataset of around 2000 nutritional labels and attempted to train our model using this dataset[7]. After training a few models with different parameters we found the model to work horribly. It was completely unable to recognize nutritional labels from the image. The dataset we used included mainly close-up images of nutritional labels where almost the entire image was continued in the nutritional label class. Additionally, the dataset seemed to contain significant numbers of nutritional labels that were not of the common format and languages found on Canadian food items, we believe this dataset is mostly European nutritional labels. Considering we were only going to have easy access to Canadian nutritional labels and taking into account the extremely poor performance of models trained with this dataset we decided to create our own dataset.

We each took several images of nutritional labels with a variety of angles and lighting conditions. In total, we had 262 images which we decided to separate into 207 training images and 55 validation images. We annotated the images using RoboFlow Annotate, a free web app that allows users to quickly annotate images with classes and then export them to formats that work with Yolo. Allocating about 80% of our images to training and 20% to validation images. The training dataset is used to build the AI model and the validation dataset is to calculate loss and do backpropagation to adjust the weights of the model. Our custom datasets worked significantly better and we were now able to reliably crop images. (the performance numbers will be discussed in the results section)

For fine-tuning the AI we ran several experiments on YOLOv5 trying different arguments to the training process. YOLOv5 offers several types of models with different weights and numbers of nodes. We experienced instability with some of Yolo's larger models and so were only able to try the YOLOv5n and

YOLOv5s models. We trained on 75 epochs with a batch size of 16 and with images resized to 1000x1000 for training. The following hyper-parameters were used which are the defaults for training in Yolo:

Figure 1:

```
lr0: 0.01
  lrf: 0.01
  momentum: 0.937
  weight_decay: 0.0005
  warmup_epochs: 3.0
  warmup_momentum: 0.8
  warmup_bias_lr: 0.1
```

Yolo uses stochastic gradient descent as its optimizer algorithm. For a detailed list of parameters on how the model was created check in comp-4102-project/models/exp5/opt.yaml.

For the OCR stages, two online sources were followed that dealt with rotating images [5][6]. The sources follow different methods for their use cases however both work by finding the angles of rotated lines in the input image, getting the median angle, and then rotating the image using that angle. The approach of rotating the nutrition labels uses a combination of these approaches. The first steps in detecting the rotation angle are to convert the image to grayscale, then the image has a bitwise_not() function applied to it, which inverts its pixel values [5]. This is useful because the lines within the label are usually black, and inverting them to white will be more suitable for the later hough line transform for detection. The image is then binary thresholded, which ensures the label lines are white against a black background (Figure 2). Following the second source, the prior steps are combined with canny edge detection and hough line transform [6]. The canny edge detection is crucial in detecting further edges after the binary thresholding. Without this step, the hough line transform detects very few of the points required for computing the angle (Figure 3). The hough line transform has its minimum line length set to 100 pixels. This was a reasonable length to expect for lines within the label; it allows small lines that could potentially be detected such as lines forming words to be filtered out. The minimum line gap allowed for the Hough transform was 15 pixels, as the thresholding and canny edge detection fragmented some of the lines, and this was found to be a reasonable gap which still preserved many of the important lines. The hough line transform returns the x and y coordinates of the 2 ends of each line. Using these points, the angle can be calculated by finding the arctangent of the slope using the coordinates of the two points [6].

The lines which the program targets are specifically the lines that separate each individual nutrition fact within the label, as their angle is the one that needs to be corrected. Certain types of lines needed to be ruled out in edge cases, for example, if a picture is given to the program relatively straight already, then the presence of straight vertical lines at 90 degrees (from the border of the label) could impact the rotation if for some reason the lines within the label are not detected. For this reason, angles from 87 to 90 degrees are not taken. Of course, this potential issue could happen in cases where the image is rotated and the 90 degree lines are still perpendicular to the label lines, however this at least rules it out in cases where the image is already aligned. From the test images used, this issue did not arise. Overall, when performing testing with the binary thresholding and without, it was found that running canny and subsequently hough line transform without thresholding was able to detect more lines and subsequently angles. However, when used on images with smaller rotation skew, including binary thresholding seems to be able to better detect those lines. Due to this, it was included in the final version, so that even smaller angles could be accounted for. Before the image is given to the OCR engine, it is converted to grayscale and then thresholded, which enhances the letters and in turn helps with OCR recognition.

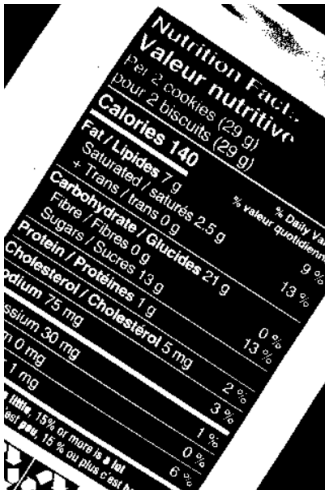
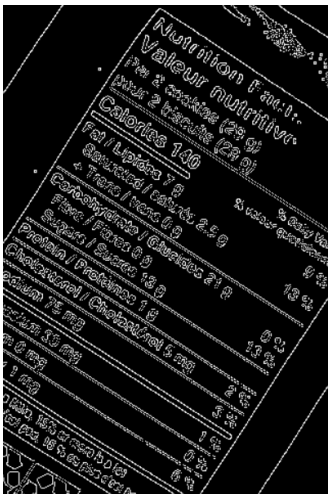
Figure 2: 

Figure 3: 

Results:

The following set of graphs (Figure 4) shows performance on the training and validation data sets for identifying the bounds of a nutritional label from an image. The top row of graphs contains data from performance on the training dataset and the bottom row shows performance on the validation dataset. The right 4 graphs display metrics about the performance of the dataset. The x-axis represents the number of epochs which all go from 0-75. Our final values on epoch 75 were: a training box loss of 0.011067 and a validation box loss of 0.0064552. A training objective loss of 0.0068638 and a validation objective loss of 0.0023613. Our losses are actually lower on validation than training which is good and means we are not overfitting the training data and our model extends well to unseen images. It can be tricky to identify what some of these statistics mean but we will go through each one and give a rough description.

Box loss: The Box loss is the Mean Squared Error and represents how far off the created bounding box is from the ground truth label bounding box [1]

Objective loss: The objective loss is the accuracy of the model in detecting the presence of the class at all[1]

Classification loss: This is not relevant because we only have one class we are looking for, therefore the model is 100% accurate in predicting class.

Precision: This is not as important of a figure in our case and measures how likely the predictions are correct in identifying the existence and lack of existence of a class. [1]

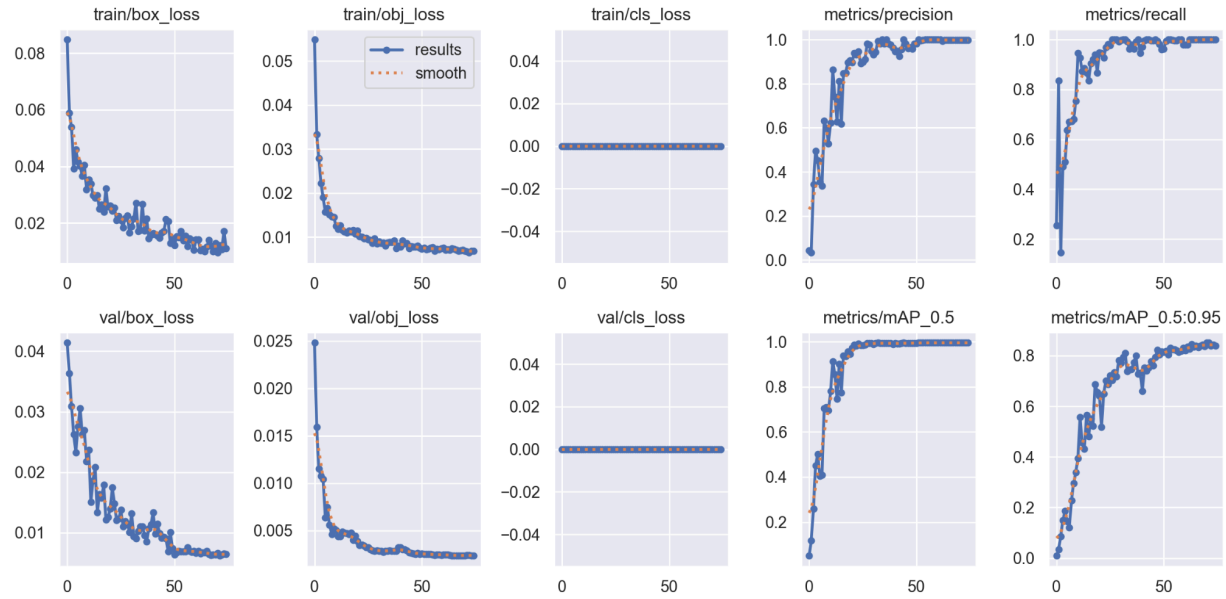
Recall: This is like precision except it only looks at cases where the object was in the image.[1]

mAP_0.5/mAP_0.5:0.95: This data is called the AveragePrecision at the Intersection of Union threshold of either 0.5 or between 0.5 and 0.95. Its definition is a bit more complicated but it is essentially a validation of what percent of the predicted bounding box overlaps with the ground truth bounding box. In the case of mAP_0.5, it is a measure of in cases where we predicted a bounding box correctly what percentage of our bounding boxes had an overlap between ground truth and predicted bounding boxes of at least 50% of their area. [2] [3]

This metric along with Box loss are the most important for our experiment since we want a precise bounding box over the nutritional label that is also well placed in order to crop it very sharply and perform OCR well. Looking at the graphs we see almost a 1.0 mAP_0.5 and a mAP_0.5:0.95 of 0.85

(0.85068 to be exact). Meaning in approximately 15% of the cases we had >= 95% overlap of area among the ground truth label and our predicted bounding box and in 99482% of cases we had at least a 50% overlap in area. The training data can be found at: comp-4102-project/models/exp5/results.csv

Figure 4:



When performing tests on the finished product, we had measured the accuracy of the application by taking photos of nutrition labels and varying them based on a few variables. For rotation, we took photos with either no rotation, a slight rotation, or a severe rotation. For lighting, photos were either categorized as having bad or good lighting. These variables were each paired with each other, making for a total of 6 possible combinations to be measured. The way accuracy is measured is by how many of the individual nutrition facts the application can recognize correctly, divided by the total number of nutrition facts, which is 13. Figure 6 shows an example of how this data was collected in a table. The full table can be seen in OCR_Accuracy.xlsx. Once all these accuracies were measured, they were totalled and averaged based on what combination of independent factors were used for that photo. The accuracies are plotted in Figure 5. From the results, it can be seen that accuracy is best when there is no rotation, and especially when there is good lighting. As for the dropoff in accuracy in rotation, it seems that once slight rotation is introduced, accuracy decreases by around 30%, however, the rate of decrease is not as extreme for further rotation. In good lighting, with slight and extreme rotation, the model is still able to get 52-55% of the label accurately.

Figure 5:

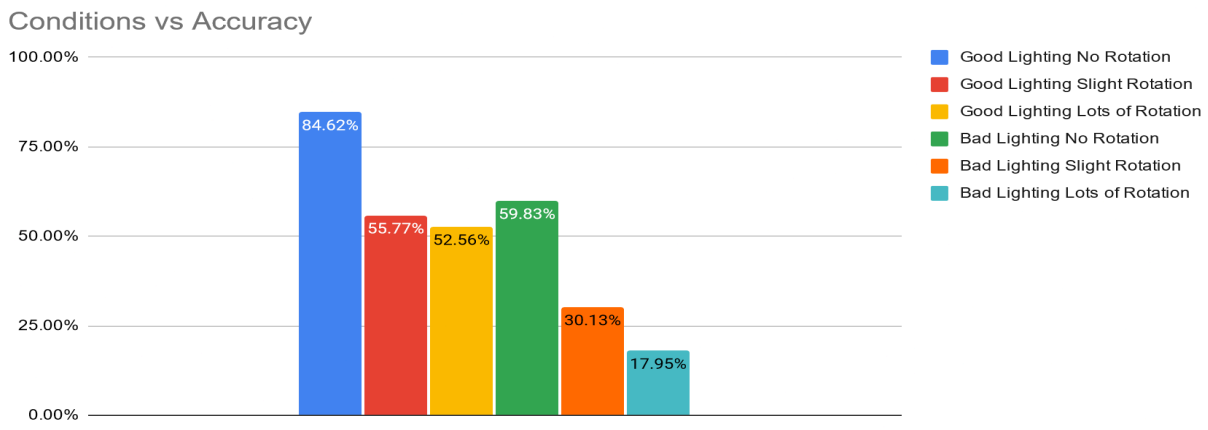


Figure 6:

Name	Correct	Total	Accuracy	Lighting, good/bad	Tilt (no, slight, lots)
rotation 3	13	13	100.00%	good	no
nutrition 2	12	13	92.31%	good	no
nutrition 1	12	13	92.31%	good	no
1000003788	7	13	53.85%	bad	slight
1000003787	9	13	69.23%	bad	no

Conclusion:

The results show a promising baseline for an application that can be used to have nutritional information digitized instantly from a photo. The combination of YOLOv5, OCR, and back-end server parsing form an application that can properly identify 85% of the nutrition information when the photo conditions are good. The decreasing accuracy seen as a result of the diminished photo quality (rotation, lighting) highlights the importance of future optimizations that could be added for improvements. Future work would be delving into why the rotation adjustments aren't more effective, as well as applying other computer vision concepts to ameliorate the quality of photos with bad lighting.

References:

[1] L. Arie, "The practical guide for Object Detection with YOLOv5 algorithm" [Online] *towards data science,* Mar. 2022. Available: https://towardsdatascience.com/the-practical-guide-for-object-detection-with-yolov5-algorithm-74c04aac4843

[2] A. Gad, "Evaluating Object Detection Models Using Mean Average Precision (mAP)" [Online] *Paperspace*, Oct. 2020. Available: https://blog.paperspace.com/mean-average-precision/

[3] J. Huid, "mAP (mean Average Precision) for Object Detection" [Online] *Medium*, Mar. 2018. Available: https://jonathan-hui.medium.com/map-mean-average-precision-for-object-detection-45c121a31173

[4] A. Kesari and J. Noel, "Nutritional Assessment" [Online] *National Library of Medicine*, Apr. 2023. Available: https://www.ncbi.nlm.nih.gov/books/NBK580496/

[5] "How to Deskew Scanned Documents" [Online] *Dynamsoft*, Oct. 2023. Available: https://www.dynamsoft.com/codepool/deskew-scanned-document.html

[6] Y. Gadade, "Automatic detection of image tilt angle" [Online], Aug. 2019. Available: https://github.com/YogeshGadade/Deep-Learning/blob/master/End_to_end_Auto_Image_tilt_angle_detection_and_correction.ipynb

[7] college, "table detection Dataset" [Online] Roboflow, Jan. 2023. Available: https://universe.roboflow.com/college-yj5l0/table-detection-xfu6w