# Explore YOLO on Food-101 Dataset

Alex Zhang
Human
Earth
https://github.com/AlexHtZhang

Cassie Huang
Human
Earth
https://github.com/cassieHuanghahaha

## Abstract

*You only look once (YOLO) [6] is a state-of-the-art, real-time object detection system. On a Titan X it processes images at 40-90 FPS and has a mAP on VOC 2007 of 78.6 % and a mAP of 48.1 % on COCO test-dev. Food classification and detection is a challenging topic due to the highly unstructured and changeable property of the food. Non-record was shown in applying YOLO on Food 101 dataset yet.*

*In this paper we explore the possibility in using the YOLO (both v1 and v2) on the Food 101 dataset [1], a challenging data set consisting of 101 food categories, with 101'000 images, 1000 images for each class.*

*Due to the limited computation power and tremendous cost on manually labeling the bounding box of the ground truth on the dataset, 400 out of 1000 randomly selected images of hamburger class and dumpling class were chosen for training. In addition, training on some highly structured object like stop-sign has been done in the same configuration for further evaluating YOLO. The training was performed for 3000 iterations for each class. The result shows that, in general YOLO prefers highly structured object in detection and YOLO v1 performs badly on small object detection compared with YOLO v2. Further, the possibility in using YOLO v2 on food classification and detection is validated in this paper. We succeeded in detecting dumpling and hamburger on images. The code is open sourced on ourGithub.*

## 1. Introduction

YOLO, a general purpose object detection which is fast, accurate, and able to recognize a wide variety of objects. However, it's only been applied and proved in detecting some relatively structured objects like car and person with a relative long time training on some relative huge training datasets (compared with what we did in this paper) [5] [6]. We would like to explore the generality of the YOLO on some food detection with a relative small dataset and short training time. Food classification and detection is a challenging topic due to the highly unstructured and changeable property of the food. Non-record was shown in applying YOLO on Food 101 dataset yet.

The dataset we focused on in this paper, Food 101 dataset [1], a challenging data set of 101 food categories, with 101'000 images, 1000 images for each class. The images were not cleaned, and thus still contain some amount of noise. This comes mostly in the form of intense colors and sometimes wrong labels. We performed training and detection on Food 101 dataset based on the YOLO v1 and YOLO v2.

We would like customize the detection to scale to level of object classification. However, labeling images for detection is far more expensive than labeling for classification or tagging(tags are often user-supplied for free) [6]. Thus we are unlikely to use the all the images for each class to detect in the Food 101 dataset. Rather, 400 out of 1000 randomly selected images of hamburger class and dumpling class were chosen for manually labeling the bounding box of the ground truth on the dataset for training. We also trained on the stop-sign dataset in the same configuration for further exploring YOLO's preference on the structure of detected object.

We trained all networks 3000 iterations and tested each of them using a testing set consisting of 30 images for a specific class. The results illustrates YOLO performs better on highly structured objects such as stop-sign.

## 2. Background

In this section, we discuss some details of original YOLO, advantages of YOLO, YOLO v2 and related works.

### 2.1. Original YOLO

You only look once (YOLO) is a system for detecting objects on the Pascal VOC 2012 dataset. It can detect the 20 Pascal object classes: person, bird, cat, cow, dog, horse, sheep, aeroplane, bicycle, boat, bus, car, motorbike, train, bottle, chair, dining table, potted plant, sofa and tv/monitor.

All prior detection systems repurpose classifiers or localizers to perform detection. They apply the model to an image at multiple locations and scales. High scoring regions of the image are considered as detections. YOLO uses a totally different approach. YOLO applies a single neural network to the full image. This network divides the image into regions and predicts bounding boxes and probabilities for each region. These bounding boxes are weighted by the predicted probabilities [5], the architecture of original YOLO can be seen in Figure 1.

### 2.2. Advantages of YOLO

YOLO has several advantages over classifier-based systems. It looks at the whole image at test time so its predictions are informed by global context in the image. It also makes predictions with a single network evaluation unlike other systems like R-CNN which requires thousands for a single image. This makes it extremely fast, more than 1000x faster than R-CNN and 100x faster than Fast R-CNN [5].

### 2.3. YOLO v2

Compared with the original YOLO, YOLO v2 uses a few tricks to improve training and increase performance. Like OverFeat and SSD, YOLO v2 uses a fully-convolutional model, but it is still trained on whole image rather than hard negatives. Like Faster R-CNN, YOLO v2 adjusts priors on bounding boxes instead of predicting the width and height outright. However, YOLO v2 still predicts the x and y coordinates directly [6].

### 2.4. Related works

Even though there were not records showing using YOLO on food detection. There were some works done on using YOLO in customized object detection [4] , [3] and there are some projects done for food detection and classification [2] , [1]. Those previous attempts either focus on applying YOLO on the highly organized dataset, or applying non-YOLO method in Food 101 dataset detection or classification. This is why we are interested in exploring the capability of YOLO on challenging Food 101 dataset on complex food detection based on YOLO.

## 3. Approach

In this section, we talk more about Data Collection, Data Annotation, Convert Data Annotation, Training List & Testing List, Code Modification and Training Configuration.

### 3.1. Data Collection

We downloaded Food 101 dataset from the following website: Food-101 [1], a challenging data set of 101 food categories, with 101'000 images, 1000 images for each class. Due to the limited computation power on the cluster and tremendous costs on manually labeling the bounding box of the ground truth on the dataset, 400 out of 1000 randomly chosen images of hamburger class and dumpling class were selected for training. In addition, in order to train some highly structured object like stop-sign for a further comparison, we got the labeled stop-sign dataset with proper annotation for YOLO from this website: Stopsign Dataset. This is a huge relief for us to save time in manually labeling the stop-sign data. The stop-sign dataset is claimed to be collected from Google image and manually annotated by the author Guanghan Ning. Special thanks to Guanghan Ning for sharing the dataset to YOLO community. Again we selected 400 images of stop-sign class for later training.

### 3.2. Data Annotation

To get the training data prepared with proper ground truth bounding box annotation, we manually labeled the 400 randomly selected images of hamburger class and dumpling class from Food 101 dataset by using the following software BBox-Label-Tool. The software will output the annotation file in .txt under the following format for each image:

```
class_number
box1_x1 box1_y1 box1_width box1_height
box2_x1 box2_y1 box2_width box2_height
...
```

However, this is not a readable format for YOLO. We need further process the annotation files by converting them into the following YOLO annotation format:

```
class_number b1_x1_r b1_y1_r b1_w_r b1_h_r
class_number b2_x2_r b2_y2_r b2_w_r b2_h_r
...
```

where 'b' is 'box', 'w' is 'width', 'h' is 'height' and 'r' is 'ratio'. The conversion was done by 'convert.py' a modularized/path independent python script written on our own. Further example can be seen in our project repository on GitHub.

### 3.3. Training List & Testing List

Training List & Testing List should follow the following format:

```
data/obj/img1.jpeg
data/obj/img2.jpeg
data/obj/img3.jpeg
...
```

where data is a folder under the parent folder 'darknet', obj is the object we want to detect, the class name and 'img' are the training/testing images for this object/class. Generation of Training List & Testing List is also done by our own script 'convert.py' mentioned above in 'Data Annotation' sub-section.

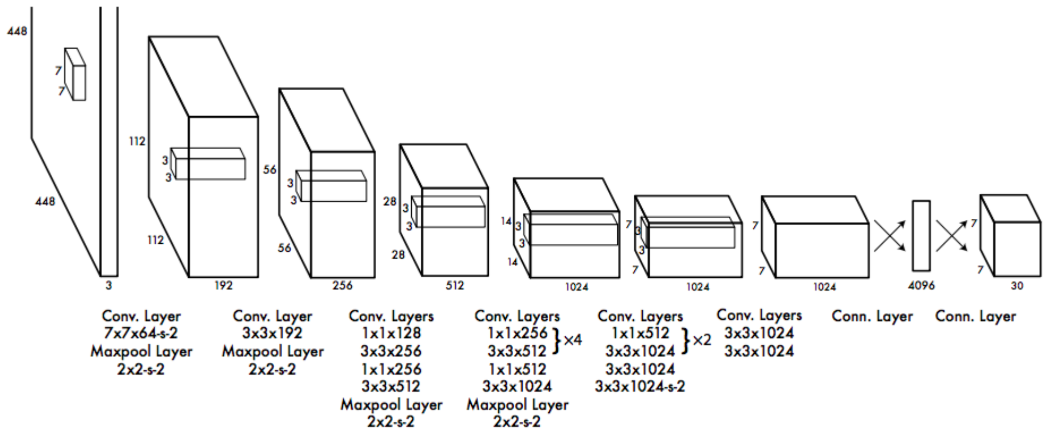

Figure 1. The Original YOLO Architecture. Detection network has 24 convolutional layers followed by 2 fully connected layers. Alternating 1 1 convolutional layers reduce the features space from preceding layers. We pre-train the convolutional layers on the ImageNet classification task at half the resolution (224 224 input image) and then double the resolution for detection.

## 3.4. Code Modification & Configuration

Since the YOLO v1 is deprecated by the author and there are many issues with our customized training on YOLO v1 (will be mentioned in experiment section), we only introduce code modification based on YOLO v2. The following are code modification and configurations we used for our YOLO v2 training. All the configurations below are experimental result from the failure training on YOLO v1 with the combination of studying from YOLO paper/YOLO community[6],

```
Create file yolo-obj.cfg with the same
content as in yolo-voc.2.0.cfg
```

where '.cfg' files should be in the 'cfg' folder under parent folder 'darknet'. Copying 'yolo-voc.2.0.cfg' to 'yolo-obj.cfg' also works. The following configuration is based on the suggestion from the YOLO Google group and the paper [6] for general customized training.

```
Change line batch to
batch=64

Change line subdivisions to
subdivisions=8

Change line classes=20 to our number
of objects

Change line #237 from
filters=125 to: filters=(classes + 5)x5.
```

So if classes=2, the number of filters for the last convolutional layer should be 35. If classes=1, it should be filters=30. Generally the number of filters depends on the classes, num and coords, i.e. equal to (classes + coords + 1)*num, where num is number of anchors). For example, for 2 objects, file 'yolo-obj.cfg' should differ from yolo-voc.2.0.cfg in such lines:

```
[convolutional]
filters=35

[region]
classes=2
```

Create file obj.names in the directory:

```
\\darknet\\data\\
```

with objects names - each in new line Create file obj.data in the directory

```
\\darknet\\data\\
```

where classes = number of objects and it should contain the following content:

```
classes= 1
train = data/train.txt
valid = data/test.txt
names = data/obj.names
backup = backup/
```

where the 'classes' is the number of classes to detect, 'train' is the path of the training list, 'valid' is the path of the testing list, 'names' is the path of the classes names, and 'backup' is where the generated weights will be stored.

One of the downsides of the YOLO project is the author didn't make good document on how to read and reuse the code. Thanks to the YOLO community:YOLO Google group, we are able to learn how the YOLO code works and customize our own configurations.

## 3.5. Training

We used pre-trained weights for the first 23 convolutinoal layers from Joseph Chet Redmon. We started training the training dataset by typing the following code:

```
./darknet detector train data/obj.data
cfg/yolo-obj.cfg pre-trained.weights
```

The key is when to stop training. This part is explained in details in the GitHub of AlexeyAB as the following:

> For a more precise definition when you should stop training, use the following manual: During training, you will see varying indicators of error, and you should stop when no longer decreases 0.060730 avg:

```
Region Avg IOU: 0.798363, Class: 0.893232,
Obj: 0.700808, No Obj: 0.004567, Avg Recall:
1.000000, count: 8

Region Avg IOU: 0.800677, Class: 0.892181,
Obj: 0.701590, No Obj: 0.004574, Avg Recall:
1.000000, count: 8

9002: 0.211667, 0.060730 avg, 0.001000 rate,
3.868000 seconds, 576128 images Loaded:
0.000000 seconds
```

For the 9002 and the 0.060730 avg:

```
9002 - iteration number (number of batch)
0.060730 avg - average loss (error)
```

> The lower average loss, the better When you see that average loss 0.xxxxxx avg no longer decreases at many iterations then you should stop training. Once training is stopped, you should take some of last .weights-files and choose the best of them: For example, you stopped training after 9000 iterations, but the best result can give one of previous weights (7000, 8000, 9000). It can happen due to overfitting. Overfitting - is case when you can detect objects on images from training-dataset, but can't detect ojbects on any others images. You should get weights from Early Stopping Point in Figure 2.

In order to get weights from Early Stopping Point. Choose weights-file with the highest IoU (intersect of union) and mAP (mean average precision). The detail explanation of IoU and mAP can be found in the following description and Figure 3 for visualization on precise, recall, IoU.

> IoU (intersect of union) - average instersect of union of objects and detections for a certain threshold.

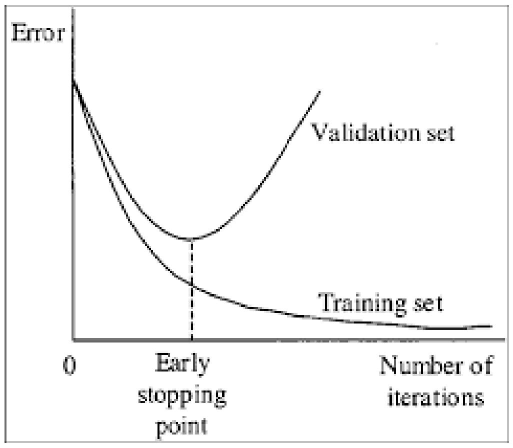

Figure 2. Use the weights near by the Early Stopping Point

> mAP (mean average precision) - mean value of average precisions for each class, where average precision is average value of 11 points on PR-curve for each possible threshold (each probability of detection) for the same class.

The data of IoU and mAP can be reached by call the internal function on the trained weights.

We trained each network using a batch size of 64, a subdivision size of 8, a momentum of 0.9 and a decay of 0.0005. The learning rate is 0.0001.

Due to the limited scope of time and computation power, we assumed the network is trained well within 3,000 iterations for each dataset, which also ensures a good comparison among these three datasets.

## 4. Experiments

The experiment was initially done on YOLO v1, the original YOLO proposed by the author, but as our training data contains many groups of small objects (a pack of dumplings) as well as object/bounding boxes in the edge of the image. There are a lot 'NaN' for the average IoU during the training process, which completely ruins the generated weight. This is a known limitation, which is already addressed in the original YOLO paper as the following:

> YOLO imposes strong spatial constraints on bounding box predictions since each grid cell only predicts two boxes and can only have one class. This spatial constraint limits the number of nearby objects that our model can predict. Our model struggles with small objects that appear in groups, such as flocks of birds. Since our model learns to predict bounding boxes from data, it struggles to generalize to objects in new or unusual aspect ratios or configurations. Our

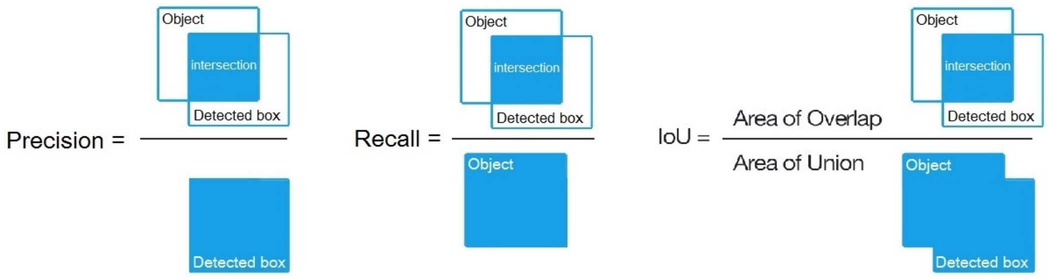

Figure 3. Visualization on precise, recall, IoU (intersect of union).

model also uses relatively coarse features for predicting bounding boxes since our architecture has multiple downsampling layers from the input image. Finally, while we train on a loss function that approximates detection performance, our loss function treats errors the same in small bounding boxes versus large bounding boxes. A small error in a large box is generally benign but a small error in a small box has a much greater effect on IOU. Our main source of error is incorrect localizations [5].

The author claimed that those issue were partially solved in the YOLO v2 by using some tricks[6]. The comparison between YOLO v1 and YOLO v2 is already given in the section 2.3.

The following sections focus on the YOLO v2, the Conduct Experiment, Evaluation Matrices, Loss Function and Results.

## 4.1. Dataset

We used three training datasets, each containing 400 images of stop-sign, dumpling and hamburger respectively. The details about collecting these datasets have been discussed in the section 3.1. For testing datasets, we download 30 images for each class from Google website. All the testing images are annotated in the same way the training images did using BBox-Label-Tool and converted into the YOLO readable form with 'convert.py'.

## 4.2. Evaluation Matrices

We use averaged loss and AP(averaged precision) to measure the performance of our networks. During training process, we can get the averaged loss from the last iteration for each network. The results will be discussed in the section 4.4. We would like to use VOC 2007 error analysis for the measurement of AP but we cannot because our datasets are not standard VOC format. Instead, we roughly picked 11 points of recalls(from 0 to 1) and obtained the corresponding precisions by changing the thresholds during the testing process. We then rank the recall and precision pairs with the ascending recalls. AP is measured by the sum of multiplication of precision and the change in recall for each point:

$$\sum_{k=1}^{N} P(k)\Delta r$$

where N is the total number of recall points, P(k) is the precision at a cutoff at k point, and $\Delta r$ is the change in recall that happened between cutoff k-1 and cutoff k. The comparison of each network with details can be found in the section 4.4.

$$\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[ (x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right]$$
$$+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[ \left(\sqrt{w_i} - \sqrt{\hat{w}_i}\right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i}\right)^2 \right]$$
$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i\right)^2$$
$$+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i\right)^2$$
$$+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2$$

Figure 4. Loss Function of YOLO v2

## 4.3. Loss Function

The loss function is the same as applied in [5] shown in Figure 4.where 1 obj i denotes if object appears in cell i and 1 obj ij denotes that the jth bounding box predictor in cell i is responsible for that prediction. It is worthy to notice that the loss function only penalizes classification error if an object shows up in that grid cell as told in [5].

| Dataset | Averaged Loss |
|---|---|
| Stop-Sign | 0.2697 |
| Dumpling | 3.4022 |
| Hamburger | 0.7180 |

Table 1. Averaged Loss for Each Dataset.

## 4.4. Result

We will discuss the results of averaged loss, precision-recall curves and averaged precision for each network and compare their performance in the following sections. Examples of the object detection are included in the section 4.4.3.

### 4.4.1 Averaged Loss

When we were training the datasets, we limited the iterations to 3,000 and recorded the averaged loss for each training. The results are shown in Table 1.

We know that the lower the averaged loss means the better the performance of network. Based on Table 1, averaged loss for Stop-sign dataset is the lowest, followed by Hamburger and averaged loss for Dumpling is the highest. It is surprised to notice that the Stop-sign dataset can achieve around 0.26 averaged loss which was only trained for 3,000 iterations. We think the reason is that stop-sign is a relatively structured object and able to be considered as a two-dimension object due to the thinner depth. The main reason that for the largest average loss on Dumpling is there are more than one object in a single image.

### 4.4.2 Precision-Recall Curves and Averaged Precision

The precision-recall curves for each dataset are shown in Figure 5. We also calculated the averaged precision provided in Table 2.The AP is not compatible with the mAP on VOC 2007 of 78.6%. It is defined for the purpose of comparing the performance of each network trained on our customized stop-sign, dumping and hamburger datasets. The stop-sign model gets the highest AP, followed by the dumpling model and the hamburger has the lowest AP of 86.46%. The reason for the relatively worse performance is the difficulty of the datasets. Compared with stop-sign and dumpling datasets, the hamburger dataset contains many variations of regular hamburgers. It might be hard to conclude a general shape of hamburgers. There is no wonder that the stop-sign model achieves the highest AP since stop-signs are highly structured objects and do not have many variations. The invariance on dumpling dataset is between these two.

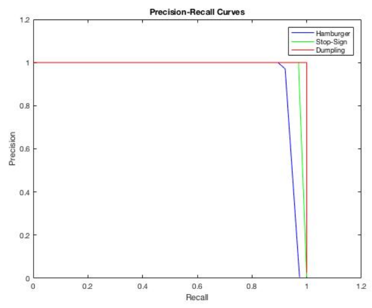

Figure 5. Precision-Recall Curves: Stop-Sign, Dumpling & Hamburger

| Dataset | Averaged Precision |
|---|---|
| Stop-Sign | 96.73% |
| Dumpling | 96.52% |
| Hamburger | 86.46% |

Table 2. Averaged Precision for Each Dataset.

### 4.4.3 Qualitative Results

We are also interested in the visualization of the testing results. We test our networks trained on each dataset and results are shown in Figure 6. The stop-sign, dumpling and hamburger images randomly collected from the websites can be successfully detected by different detectors.

But there are also some test results with unexpected mistakes as shown in Figure 7. We can see the model treats the bus front light as a stop-sign and the blurred background is considered as a hamburger. The bus front light is red and round, and has some white spots in the center, which is easily to be considered as a stop-sign based on the analysis of the key feature of many original stop-signs. We suppose this is caused by the limited training times. We only train the network for 3,000 iterations. The mistake made by the hamburger detector is due to the training data annotation process. There are many different types of hamburgers in the dataset. Some of them show the internal structures while some show the external structures. This hamburger dataset is hard to train on YOLO which models the size and shape of objects.

## 5. Conclusion

Through our experiments, as addressed in the original paper, the original YOLO has many issues in training and detecting small objects , a pack of objects and object at the edge. Meanwhile the YOLO v2 partially solved those issue.

We validate the possibility of using YOLO v2 to detect food by using Food 101 dataset[1]and discuss the influence of the shape of objects on the performance. We find that the YOLO v2 is good at detecting highly-structured objects. The stop-sign model outperforms other two models. However, wrong detections still exist. Better performance

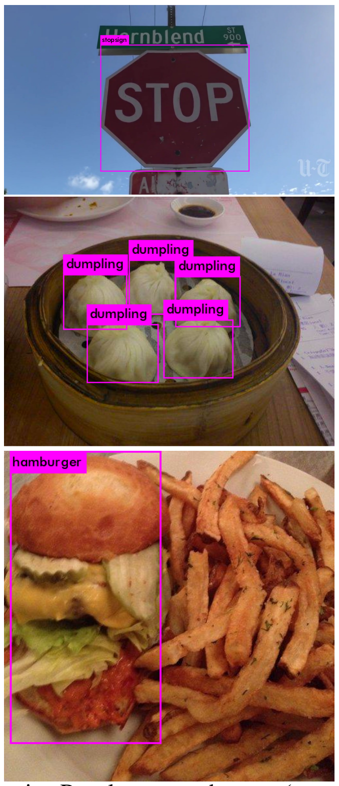

Figure 6. Qualitative Results: top to bottom (stop-sign, dumpling and hamburger)

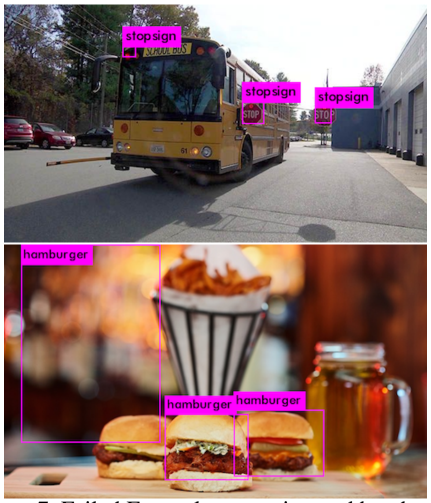

Figure 7. Failed Example: stop-sign and hamburger

could be obtained by increasing the number of iteration and epoch and by additional augmentation, such as flipped and cropped, during training procedure. But due to the limited scope of time and computation power, we leave this to future work. We are also interested in developing a real-time system for providing calories contained in a meal by applying YOLO v2 to detect food categories and using the data combination method in [6] to associate the detected food name with its corresponding calories. This can be done in the future.

## References

[1] L. Bossard, M. Guillaumin, and L. Van Gool. Food-101 – mining discriminative components with random forests. In *European Conference on Computer Vision*, 2014. 1, 2, 6

[2] T. Brosnan and D.-W. Sun. Improving quality inspection of food products by computer vision—-a review. *Journal of food engineering*, 61(1):3–16, 2004. 2

[3] Q. Peng, W. Luo, G. Hong, M. Feng, Y. Xia, L. Yu, X. Hao, X. Wang, and M. Li. Pedestrian detection for transformer substation based on gaussian mixture model and yolo. In *Intelligent Human-Machine Systems and Cybernetics (IHMSC), 2016 8th International Conference on*, volume 2, pages 562–565. IEEE, 2016. 2

[4] X. Qiu and S. Zhang. Hand detection for grab-and-go groceries. 2

[5] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016. 1, 2, 5

[6] J. Redmon and A. Farhadi. Yolo9000: Better, faster, stronger. *arXiv preprint arXiv:1612.08242*, 2016. 1, 2, 3, 5, 7