Hand Detection For Grab-and-Go Groceries

Xianlei Qiu
Stanford University
xianlei@stanford.edu

Shuying Zhang
Stanford University
shuyingz@stanford.edu

Abstract

Hands detection system is a very critical component in realizing fully-automatic grab-and-go groceries. In this project, we propose to implement a near real-time hand detector using You only look once(YOLO) v2 network. The model we trained on egohand dataset can achieve test mAP of 90.53 with prediction speed of 30 frames/s on Nvidia Tesla K80 GPU.

1. Introduction

Recent advancements in computer vision have made the grab-and-go grocery stores like Amazon Go a reality. Shoppers can simply walk in the store, grab the items and walk out, without waiting in a long line for checkout. The automation of the checkout process relies heavily on computer vision systems capable of tracking items grabbed by a certain customer.

Hands detection is a key component in such an intelligent system. Customers' behaviors during shopping are complex and hard to predict. For instance, it is common for a shopper to grab items from shelf and later put them back. To further understand customers' behaviors, it is extremely important to detect and track their hands in video streams.

To tackle the challenges from complex shopping environment in real world, an ideal hand detector should not only provide high-accuracy prediction, such a prediction should also be made in a reasonable short time to meet real-time requirement. Besides the accuracy and speed requirement, it is also practical to take computation efficiency into account. The hand detection algorithm should be light-weight so that even a low cost embedded system can complete the detection task by adopting the algorithm.

2. Related Work

Many traditional computer vision methods have been proposed to detect hands in an image. The frequency spectrum analysis detects hands by categorizing hand-look-like frequency features [7]. Another approach is to use segmentation via pixel-level information [2]. However, most of these approaches rely on detecting skin tone pixel, which could be ambiguous if faces or other skin regions are also present in the image.

Recently, the adoption of deep convolutional neural network has largely improved the performance of image classification and object detection. Object detection is more challenging as it involves proposing bounding boxes for corresponding objects.

2.1. Region Proposal Based Detectors

Regions With CNNs (R-CNN) [6] is a object detection network that relies on external region proposal system. Although significantly faster and more accurate than traditional HOG-like features based methods, RCNN still suffers from the speed performance issue inherited from the external region proposal system. In our evaluation, to propose regions for a single image, selective search [16] takes 5 seconds on a 8-core CPU. Fast R-CNN [5] reused feature maps from convolution output by projecting region proposals to a Roi Pooling layer. Compared to R-CNN, this approach achieved 25 times faster performance by avoiding repeated computation for feature maps. However, fast R-CNN still depends on external region proposals, which is the bottleneck of train and prediction speed. The newest Faster R-CNN [12] got rid of external region proposal by inserting Region Proposal Network after deep convolution layers. This improvement helped Faster RCNN gain 10 times faster speed than Fast RCNN.

All the region based detectors described above share a common characteristic: they have one part of their network dedicated to providing region proposals followed by a high quality classifier to classify these proposals. Those methods

Detection Framework	mAP	FPS
R-CNN O-Net BB [6]	66.0	0.02
Fast R-CNN	70.0	0.5
Faster R-CNN VGG-16	73.2	7
YOLO	63.4	45
SSD300	74.3	46
YOLOv2 480×480	77.8	59

Table 1. Accuracy and Speed Comparison for different models [11]. Models are trained on PASCAL VOC 2007 + 2012 datasets.

are very accurate but come at cost of over-complicated network and high computational load. In other words, they are not fit to be used in low cost embedded systems.

2.2. Single Shot Detectors

Another category of detectors attack object detection problem from a different angle, unifying the separate components of object detection into a single neural network. These algorithms only take a single shot of images and can achieve higher speed compared to region proposal based detectors.

The You Only Look Once(YOLO) [10] network predicts bounding boxes and classes of objects at the same time based on activation maps output from deep ConvNets. YOLO9000 [11] is the upgraded version of YOLO, which can detect up to 9000 classes. YOLO9000 is faster and can reach 73.4 mAP on Pascal [4] dataset, whereas original YOLO achieves 63.4 mAP. The Single Shot MultiBox Detector(SSD) [9] looked at different feature maps from convolutional layers to predict bounding boxes with different scales. SDD is most suitable for uses where scales of bounding boxes vary from a large range.

Table 1 summarizes the accuracy and speed performance of detection framework we have described. As we can see, single shot detectors tend to perform better in terms of accuracy and speed.

3. Method

When we first addressed the hand detection problems, we chose an approach similar to RCNN. We used selective search to propose 2000 regions for an image, and fed these regions into a binary classifier to see if an input region is a hand. This system also need to deal with bounding box regression. Shortly, we realized this system is extremely slow. When predicting, the region proposal stage itself took 5 seconds per image, not to mention the time consumed by the binary classifier to predict 2000 regions for one image. So, after some research and experiments, we switched gear to a more promising detector, namely YOLO. The final model

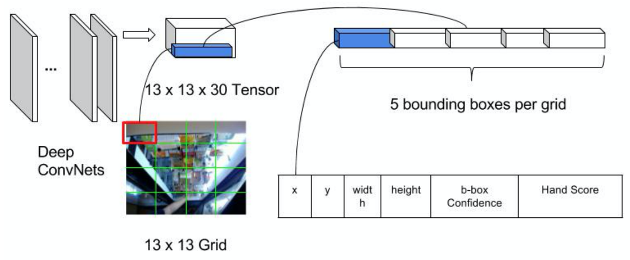

Figure 1. Explanation of YOLO Workflow. S = 13, B =5, C=1 for hand.

we used is based on a tiny version of YOLO9000. We will describe the principle of YOLO model and some optimizations implemented by YOLO9000 in the following sections.

3.1. Principle of YOLO Detectors

YOLO simply looks at the final feature map output by ConvNets and predicts bounding boxes and classes scores simultaneously. The way YOLO works as if it divides the input image to $S \times S$ cells and each cell is responsible for predicting its own bounding boxes and corresponding classes scores. As Figure 1 shows, the final output of the network is a $S \times S \times X$ tensor, where X depends on number of classes and YOLO version. Each point in the final output tensor looks at the cell in original image with the same spatial position and predicts B bounding boxes. For each bounding boxes, YOLO not only predicts x, y, width and height, but also a confidence score that represents the IOU with ground truth bounding box and probability of an object present in that cell:

$$Confidence_{b-box} = Pr(Object) \times IOU_{pred}^{truth} \tag{1}$$

Each cell also predicts C conditional class probabilities, $Pr(Class_i|Object)$. YOLO only predicts one set class probabilities for each cell while YOLO v2 predicts for each bounding boxes. The confidence score for class can be computed as:

$$\begin{aligned} Confidence_{class_i} &= Pr(Class_i|Object) \times Pr(Object) \\ &\quad \times IOU_{pred}^{truth} \\ &= Pr(Class_i) \times IOU_{pred}^{truth} \end{aligned} \tag{2}$$

From Equation 2, we can see the class scores encode both the probability of that class appearing in the box and how well the predicted box fits the object. The final prediction will be made by the predictor with the highest IOU with the ground truth bounding box. During training, YOLO optimize a multi-part loss function as Equation 3 shows. $\mathbb{1}_i^{\text{obj}}$ denotes if an object appears in cell i and $\mathbb{1}_{ij}^{\text{obj}}$ denotes that

jth bounding box predictor in cell i is responsible for that prediction. The first 2 parts in the equation penalize the loss from bounding boxes coordinate predictions. The 3rd, 4th and 5th parts in the equation penalize the loss from classification.

$$\begin{aligned}
&\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \\
&+ \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right] \\
&+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2 \\
&+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2 \\
&+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2 \qquad (3)
\end{aligned}$$

3.2. YOLO9000 Optimization

YOLO9000, a.k.a. YOLO v2, boosted mAP of YOLO from 63.4 to 78.6 on VOC2007 dataset, by adopting multiple advanced deep learning techniques. The novel strategy that jointly training on ImageNet [3] classification dataset and COCO [8] detection dataset helps YOLO v2 detect object classes that don't have labeled detection data. By adding Batch Normalization on all of convolutional layers, YOLO v2 got 2% improvement in mAP. YOLO v2 removed the fully-connected layer used by YOLO and replaced with convolutional layers to predict bounding boxes based on anchor boxes, a way similar to Faster RCNN. In this way, YOLO v2 predicts a set of class probability for each bounding box rather than only for a single cell in YOLO v1. This trick helps YOLO v2 gain 88% recall, compared to 81% for the original version.

Other improvements includes using high resolution classifier, using K-means to automatically finding good priors and multi-scale training.

3.3. Hand Detector Network Architecture

The model we used is a tiny version of YOLO9000, see Figure 2. It contains 8 convolutional layers before the final output convolutional layer. Each convolutional layer has a batch normalization layer before leaky ReLu activation and a maxpool layer after. We use S=13, B = 5 and C =1(hand), so we expect that the final convolutional layer output a tensor with shape $13 \times 13 \times 30$, where 30 = 5 bounding boxes $\times$ (4 coordinates + 1 b-box score + 1 class probability).

4. Datasets

4.1. Egohand

The EgoHands [1] dataset consists of a total of 48 Google Glass videos. Each video has 100 frames with resolution of 1280×720 and frames are annotated with ground-truth boxes of hands (Figure 3). For each video, We selected the first 70 frames for training and the last 30 frames for testing. Therefore, the total training dataset consists of 3360 images and the test set consists of 1440 images.

4.2. Grocery Video Frames

Besides Egohand dataset, we were also provided with 10 videos, which recorded customers behavior of removing food items from a self-serving fridge. For this real life use case, only detecting hands seems not enough to tracking customers' movement. While fetching foods, customers' hands may be under the shelf and become invisible to the camera. Therefore, it makes more sense to detect both hands and arms during a shopping events. For each video, we randomly sampled 50 frames and labeled the ground-truth boxes of hands and arms using the labelImg [15] labeling tool, as Figure 4 shown.

4.3. Data Preprocessing

Image preprocessing was conducted before feeding data to the network. The input images are of varying sizes. We first resized the image to 416x416, to be compatible with the input layer of the YOLO network. If the image has an accompanied annotation, meaning this preprocessing is serving the training process, then this image is transformed with random noise to augment training data. Scaling, translation, flipping and recolor are used to transform the image. The accompanied parsed annotation will be updated accordingly.

5. Experiments, Results and Discussion

5.1. Training

Our implementation is based on a open source YOLO framework [14]. We first tried to train our model from ground without basing on pretrained model. And we found that the model is really hard to converge and tended to reach a local optimal where it will predict nothing at all. Then we turned to resort to a model which was pre-trained on PSCAL VOC 2007 dataset.

We use RMSProp with decay rate 0.9 as our trainer because RMSProp has least overshoot effect compared to SGD+Momentum, Nesterov and Adam. In our case, model stability and converge is more important. We split our training process into 2 stages. In first stage, we use batch 16 and learning rate 1e-5 in order to quickly reduce loss. Large

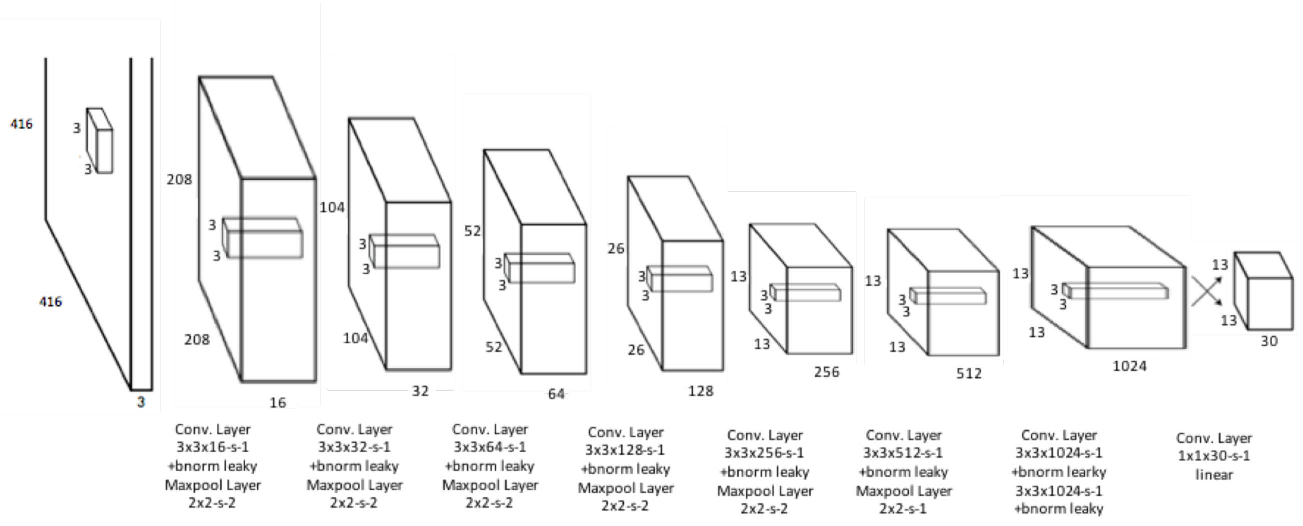
416
416
3
3
3
208
208
16
3
3
104
104
32
52
52
64
26
26
128
13
13
256
13
13
512
13
13
1024
13
13
30
Conv. Layer 3x3x16-s-1 +bnorm leaky Maxpool Layer 2x2-s-2
Conv. Layer 3x3x32-s-1 +bnorm leaky Maxpool Layer 2x2-s-2
Conv. Layer 3x3x64-s-1 +bnorm leaky Maxpool Layer 2x2-s-2
Conv. Layer 3x3x128-s-1 +bnorm leaky Maxpool Layer 2x2-s-2
Conv. Layer 3x3x256-s-1 +bnorm leaky Maxpool Layer 2x2-s-2
Conv. Layer 3x3x512-s-1 +bnorm leaky Maxpool Layer 2x2-s-1
Conv. Layer 3x3x1024-s-1 +bnorm learky 3x3x1024-s-1 +bnorm leaky
Conv. Layer 1x1x30-s-1 linear

Figure 2. Tiny YOLO v2 Architecture

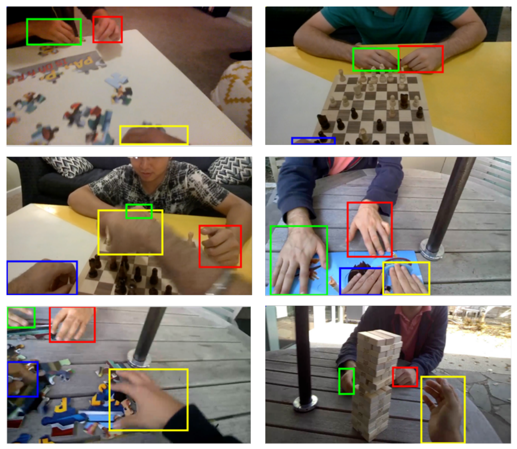

Figure 3. Samples of Egohand datasets and corresponding ground truth bounding box.

	Stage 1	Stage 2
Trainer	RMSProp	RMSProp
Momentum	0.9	0.9
Learning Rate	1e-5	1e-6
Batch Size	16	64
Epchos	20	10
Final Loss	4.32	2.98

Table 2. Summary of Training Process

batch batch size like 64 will dramatically slow down training. After training for 20 epochs, the model got stuck and the loss cannot be reduced any further. Then we change our batch size to 64 and learning rate to 1e-6 for a finer granularity training. The summary of training process is shown on Table 2.

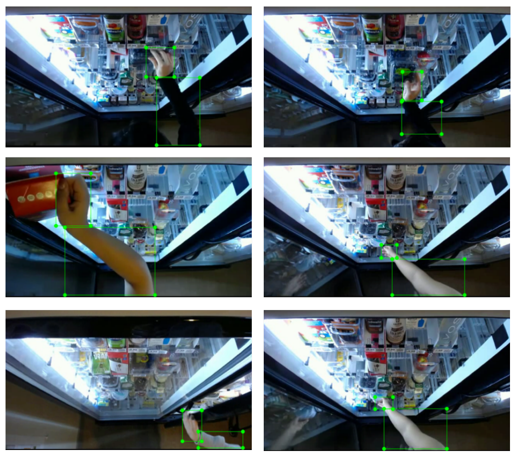

Figure 4. Samples of self-serving fridge video stream. We labeled 2 classes for each frame, hands and arms.

5.2. Accuracy and Speed Performance

Since we do not have enough time to label more data for grocery video data, we only overfit the model on 500 frames for demo purpose, see Figure 5. Therefore, in this section, we will only show the evaluation for Egohand model. While counting True Positive, we consider only one prediction is true for one hand. Other predictions overlapping with True Positive will be considered as False Positive. With IOU = 0.5, the recall vs precision plot shows in Figure 6. The mAP is calculated by averaging 11 precisions when recall is in range[0, 1] with 0.1 interval. The mAP for training dataset and test dataset are:

$$\text{Training mAP} = 91.54$$

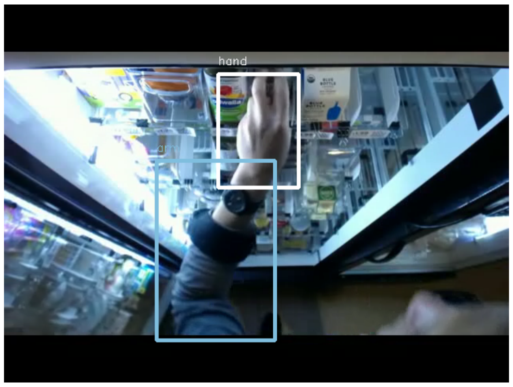

Figure 5. Fridge Image Detection for hand and arm

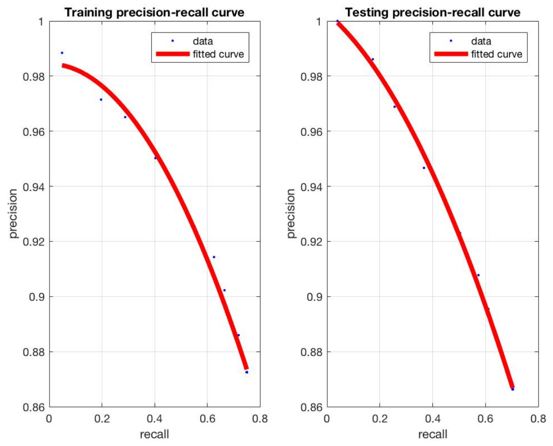

Figure 6. Recall vs Precision when IOU = 0.5. Left: Training. Right: Test

Testing mAP = 90.63

On our 8 vCPUs, 30 GB memory instance, our model can achieve 4.815 FPS. On NVIDIA Tesla K80 GPU, our model can achieve 30 FPS speed.

5.3. Weights Visualization

Figure 7 shows activations of 16 filters on the first CONV layer. We can see that some of the filters that are useful for detecting the features were activated. They are used by the network to extract attributes of the images, such as edges, texture, coarse shapes, etc. We also extracted the activation map from 19th layer, i.e. the last second CONV layer, which has a size of 13x13x1024. We showed 64 filters with the highest sum values of the filter. It is interesting to see that these filters illustrate a variety of patterns, but more or less look like the shape of hands.

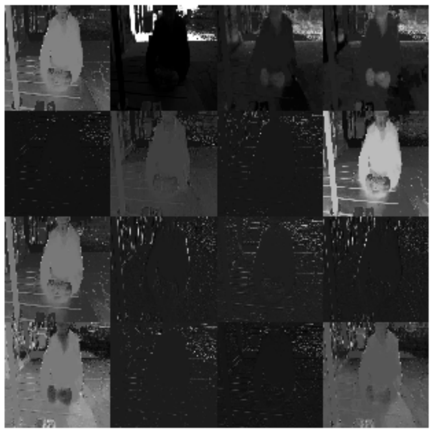

Figure 7. Visualization of activation after first Conv Layer of the tiny-YOLO-VOC-hand architecture.

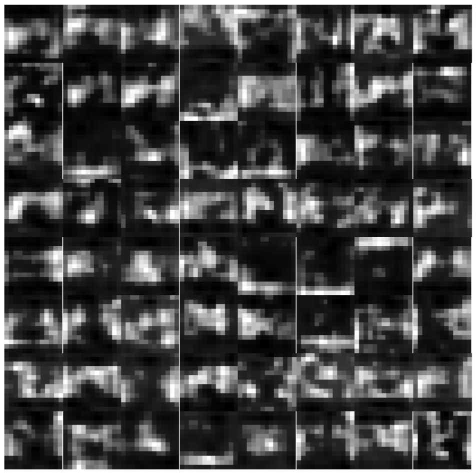

Figure 8. Visualization of activation after the last second Conv Layer of the tiny-YOLO-VOC-hand architecture

5.4. Saliency Map

Saliency map is a valuable way to visualize how the network make decisions using different image pixels. It is widely used as a visualization for deep ConvNets classification[13]. However, we would like to see if saliency can be applied to visualize ConvNets for object detection. Similar to image classification, the YOLO object detection output a class score tensor for making prediction. Unlike image classification where there is one scalar class score for each class, the YOLO looks at the whole image. We wanted to gain intuition on how the image is decomposed into structurally representative segments. We extracted the true class score in the tensor and computed a class saliency

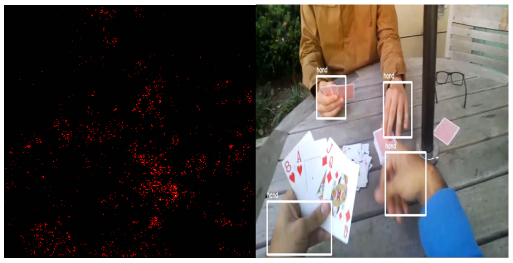

Figure 9. Saliency map of a typical image with correct prediction result.

map by computing the gradient of the class score respect to the input image. Recall that the output tensor from the YOLO network is a 13x13x30 tensor. We reshaped this tensor to a 13x13x5x(30/B) tensor, where B is 5 in our case. output[:,:,:,:5] is the (x, y, w, h, confidence), The unnormalized class score is:

$$classScore = output[:, :, :, 5 :].$$

Note that we only have one class, therefore we used this class score which was used to calculate the gradient of the image to create our saliency map. The output is a 416x416 saliency map image. Figure 9 shows the saliency map of a typical image. The predicted boxes labeled in this image match all the ground-truth boxes. We can see that the pixel values are slightly higher in the area of the predict bounding boxes, which indicates that the YOLO network actually makes decision using pixels that is more correlated with the class. Figure 10 is another example. In the saliency map, besides that the pixels are higher in the predicted box region, the pixels surrounding the flower pot on the left edge of the image are also picked up by the YOLO architecture. We think that this might be due to the fact that the color of the flower pot is close to that of hands, which might be something interesting for the convolutional layers. However, in Figure 11, the chess board also has colors that are closer to the hands but are not picked up by the YOLO architecture. This implies that the network seek things more than just the color, probably also the heuristics of the class. We also presented an example saliency map when the network fails to make the correct prediction. In Figure12, the hand picking up the chess piece is not picked up. In the saliency map, the region where that hand is located is actually higher. This indicates that the network looks for candidate regions in the image and make predictions on those regions.

6. Conclusion and Future Work

We implemented a hand detector using tiny version of YOLO9000, which can achieve 90.63 mAP on Egohand test datasets. With 30 FPS, this model can be applied in real time hands tracking tasks. The total size of our tiny model is around 250 MB so that it can be fit in low cost embedded system.

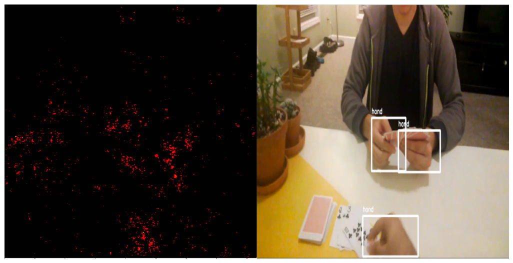

Figure 10. Saliency map of an image where some background pixels are picked up.

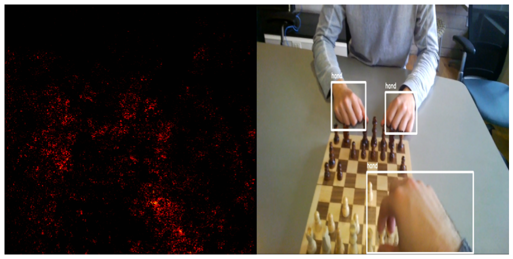

Figure 11. Saliency map of an image where background pixels with hand-alike color are not picked up.

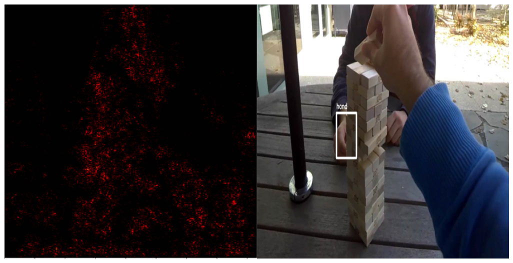

Figure 12. Saliency map of a mislabeled image where lots of background pixels are picked up.

The YOLO9000 model can be easily extended to detect other classes. Therefore, our future work should include detecting other food classes, understanding user behavior and ultimately detecting what food is fetched by customers.

References

[1] S. Bambach, S. Lee, D. J. Crandall, and C. Yu. Lending a hand: Detecting hands and recognizing activities in complex egocentric interactions. In *The IEEE International Conference on Computer Vision (ICCV)*, December 2015.

[2] Betancourt, Lpez, Regazzoni, and Rauterberg. A sequential classifier for hand detection in the framework of egocentric vision., 2014.

[3] J. Deng, W. Dong, R. Socher, L. J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, June 2009.

[4] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (voc) challenge. *International Journal of Computer Vision*, 88(2):303–338, June 2010.

[5] R. Girshick. Fast r-cnn. In *International Conference on Computer Vision (ICCV)*, 2015.

[6] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *Proceedings of the IEEE conference on computer vision and pattern recognition.*, 2014.

[7] M. Klsch and M. Turk. Robust hand detection., 2004.

[8] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollár. Microsoft COCO: Common Objects in Context. *ArXiv e-prints*, May 2014.

[9] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg. SSD: Single Shot MultiBox Detector. *ArXiv e-prints*, Dec. 2015.

[10] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You Only Look Once: Unified, Real-Time Object Detection. *ArXiv e-prints*, June 2015.

[11] J. Redmon and A. Farhadi. Yolo9000: Better, faster, stronger. *arXiv preprint arXiv:1612.08242*, 2016.

[12] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *In Advances in neural information processing systems.*, pages 91–99, 2015.

[13] A. V. Simonyan, Karen and A. Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv.*, 2013.

[14] thtrieu. darkflow. `https://github.com/thtrieu/darkflow`, 2016.

[15] tzutalin. Labelimg. `https://github.com/tzutalin/labelImg`, 2015.

[16] J. R. R. Uijlings, K. E. A. van de Sande, T. Gevers, and A. W. M. Smeulders. Selective search for object recognition. *International Journal of Computer Vision*, 104(2):154–171, 2013.