Probabilistic and Deep Machine Learning Techniques for Object Detection

Final Project Report
UCLA CS260: Machine Learning Algorithms, Winter 2021

Terri Tsai
UID: 805624292
tctsai@g.ucla.edu

Kenny Chen
UID: 505219294
kennyjchen@ucla.edu

***Abstract*— In this project, we developed and compared methods for automatic object segmentation and bounding box detection using both probabilistic and deep learning techniques. More specifically, we use both statistical-based methods and convolutional neural network models to detect and extract the bounding box of a red cup within a give image. In our probabilistic approach, we train and and classify using a single Gaussian model for each class to estimate position and orientation of the cup; in the deep-learning approach, we fine-tune a pretrained YOLOv3 model on our dataset for several epochs to influence the initial weights towards our object of interest. By developing our own object detection pipeline and comparing/contrasting these different methods on our novel dataset, we can gain further insight into the progression of machine learning algorithms for object detection and computer vision as a whole.**

***Index Terms*— machine learning, computer vision, object detection, probabilistic models, learning algorithms**

I. Introduction

Problems that are seemingly trivial for humans to accomplish are often the hardest to solve using machine learning. For example, the human brain has the ability to learn, given time and experience, to be able to recognize specific objects in an image within fractions of a second upon seeing them. Yet, when tasked to automate this process on a computer, the problem quickly becomes non-trivial. Our abilities to quickly identify objects come from our innate intuition of the world we live in and our years of experience of interacting with the environment around us; however, when programming this "intuition," we no longer have the extensive background experience to work with and thus must approach this problem more carefully. By drawing inspiration from our own biological system, we can develop our own learning algorithm to extract both low-level and high-level features of an item to provide a systematic understanding of objects programmatically.

To model this behavior, many object recognition techniques in machine learning realm have been explored and developed, with the current state of the art surrounding learning with neural networks and deep learning. More specifically, modern approaches to object identification use convolutional neural networks (CNNs) to learn a latent representation of a high-resolution image that, when fed into a series of fully-connected layers, can classify an image holistically (i.e., cat or dog). However, by removing these fully-connected layers at the end (and hence having a fully covolutional neural network), bounding boxes can be extracted which localize an object of interest in the image.

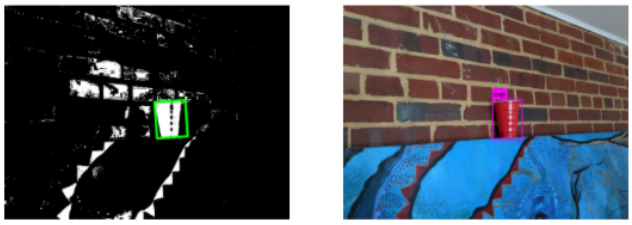


Fig. 1: **Object Detection via Probabilistic and Deep Learning Methods.** We developed two pipelines for object detection using traditional and modern machine learning approaches. Left: our probabilistic approach which accounts for image lighting invariances, performs pixel-wise classification, then detects a bounding box through a series of geometric heuristics. Right: our deep learning approach which fine-tunes a pre-trained YOLOv3 model using our novel dataset in a fully convolutional neural network architecture.

With the advent of deep learning in recent years though, it's easy to forget that deep learning is only a subset of machine learning. In this project, we explore the foundations of object detection and develop our own probabilistic machine learning approach for object localization via a bounding box. More specifically, using the tools that we learned in CS260: Machine Learning Algorithms, we develop an end-to-end statistics-based learning framework which extracts the bounding box of an object of interest (i.e., a red cup). Our method accounts for lighting invariances that can be found across different environments by training and classifying via single Gaussian models on lighting-normalized, low-level features. Afterwards, higher-level features are extracted by detecting the contours of the output mask, which subsequently is fed through a series of heuristics to extract the bounding box. We compare our method with YOLOv3 [1] using a model which we fine-tuned using own novel collected dataset. Our experiments show that, while modern

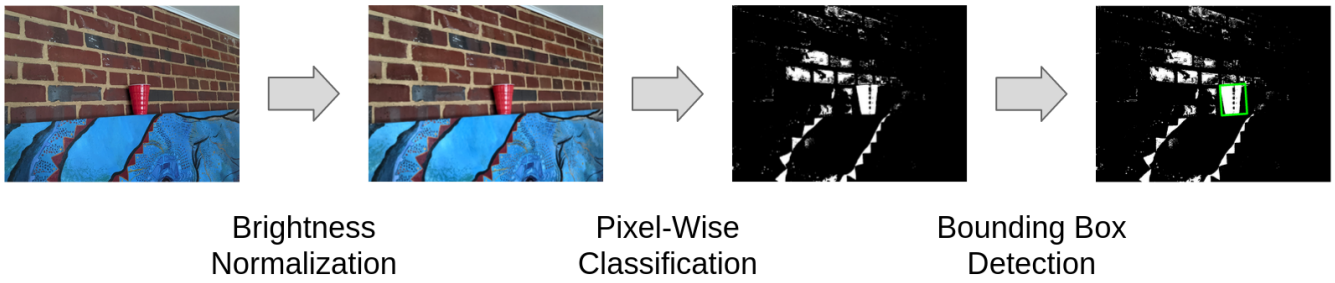


Fig. 2: **System Architecture.** System architecture for our probabilistic object detector using a traditional, statistical approach. We extract pixel-based RGB features after normalizing brightness to account for different lighting conditions. Contours and bounding boxes are then subsequently detected using geometric constraints and other heuristics to filter out noisy detections.

deep learning approaches are certainly more robust to various environments and object orientations, our transparent, probabilistic pipeline allows users to understand every step along the way, which is important for both easy improvements in performance and educational purposes.

Related Work

Object detection is one of the most fundamental and challenging problems in computer vision, and it has received great attention not only in recent years but starting from the 1990s. While modern object detection methods largely use deep learning to localize and understand a given image, there has been tremendous technical development in this field starting from the cold war era [2]. In traditional detectors, early object detection algorithms used low-level, handcrafted features partly because deep learning was not yet fully matured and partly because of computational efficiency, in which such detectors do not rely on powerful GPUs. Works such as the Viola Jones face detector [3], the Histogram of Oriented Gradients (HOG) feature descriptor [4], and the Deformable Part-based Model (DPM) [5] were significant milestones in traditional object detection and paved the way for more innovations in this area. While modern object detection architectures have far surpassed these traditional methods in terms of accuracy and robustness, the insights gained from such methods and the deep influences can be seen in how today's methods are constructed.

In the modern deep learning era, deep learning paved the way towards more robust performance by learning higher-level, semantic-like feature representations in an image. Starting from two-stage detectors such as SPPNet [6] and Faster R-CNN [7] to one-stage detectors such as [1], the usage of deep learning and the power of GPUs provided much more accurate and robust detection performance at the cost of computational complexity. Nonetheless, these methods performance far surpass those of traditional methods from the early 2000s and are definitely the foundation for what's to come next.

In this project, we explore methods from both eras and compare not only performance, but also robustness, complexity, and development process. For our traditional approach, we use color segmentation techniques as described in [8] to classify and detect via pixel features. As for our deep learning pipeline, we explore the state-of-the-art YOLOv3 architecture [1] and fine-tune a pre-packaged model to influence weights towards our novel dataset. Although the deep method quite predictably surpasses the statistical pipeline, we nonetheless gain valuable insights on how such traditional approaches are deeply influenced in modern methods.

II. Methods

In this section, we describe our data collection process, our end-to-end statistical object detection approach, and our deep-learning pipeline. Our novel dataset consists of images collected around the Maryland state, which we used for both probabilistic and deep approaches. Our probabilistic approach contains three steps, including preprocessing, pixel-wise classification, and contour detection; on the other-hand, our deep-learning pipeline uses CNNs to regress bounding boxes of objects using a trained, fine-tuned network model. We will discuss our technical approach and subsequent challenges in the following subsections.

A. Data Collection

To get the full implementation experience, we collected our own data and processed it ourselves. A total of 100 photos were taken of red cups under different lighting conditions and placements, with only one cup in each photo. Several factors we kept in mind of when taking the photos include: the entire cup should be within the frame of the photo, and if the cup were to be covered by objects, it is not covered by more than 25%. This is to ensure that our pixel detection method can work reasonably. The photos were then reformatted to be 1200 by 900 pixels *.jpg* format for computational efficiency, as this is a more reasonable size to process.

B. Probabilistic Object Detection

1) Image Lighting Normalization: Our dataset consisted of images taken in various lighting conditions (such as in direct sunlight or in dark, shadowy places), and to account for such lighting invariances, we normalized the brightness across images in both training and test sets. In particular, to do this we converted the raw red-green-blue (RGB) color images to the hue-saturation-value (HSV) color space. Whereas the RGB colorspace is a linear, 3-dimensional color space (where each axis represents the red/green/blue intensity), the

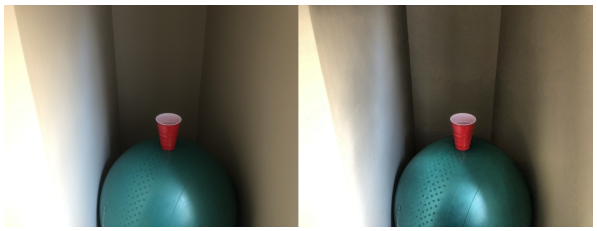


Fig. 3: **Brightness Normalization.** Left: Original image of our red cup in a shady location. Right: Image that has been adaptively histogram equalized on the value layer.

HSV colorspace is an alternative representation in which the *value* channel represents image brightness.

To normalize, we used a variant of histogram equalization, called Contrast Limited Adaptive Histogram Equalization (CLAHE), to smooth out the value channel so that brightness values were more spread out and thus more entirely represented. Whereas normal histogram equalization normalizes per-frame, CLAHE slides an $8x8$ window across the image and equalizes the pixels only within each window; this increases contrast of the image without losing a significant amount of color information. This is especially important for our project, as we are leveraging the red cup's bright red hues in order to classify and localize it.

2) Pixel-Wise Classification:

a) Machine Learning Model: Localizing our red object in noisy environments is a challenging task, and thus we must approach this problem from a machine learning standpoint. To accomplish this, we classify each pixel using a single Gaussian learning model to compute the Bayes Decision Rule (BDR) of each class for each pixel; a pixel's subsequent class is then the highest number (i.e., the highest probability) of the calculated BDR values. We chose to use a single Gaussian classifier for its simplicity in implementation and its generally-accepted high accuracy for classification tasks like the one in this project. However, we note that the classification performance could probably be improved by moving towards a multi-model Gaussian model (where each class probability is modeled by the union of several Gaussian distributions instead of just one).

More specifically, given all classes i and the testing data $\vec{\boldsymbol{x}}$, we can compute which class a specific pixel (RGB) belongs to via the Bayesian Decision Rule:

$$i^*(\vec{\boldsymbol{x}}) = \arg\max_i \left\{ P_{Y|X}(i|\vec{\boldsymbol{x}}) \right\} ,$$

or, the class that maximizes the probability of class i given data vector $\vec{\boldsymbol{x}}$. We can simplify these terms using Bayes' Theorem and get:

$$i^*(\vec{\boldsymbol{x}}) = \arg\max_i \left\{ \frac{P_{X|Y}(\vec{\boldsymbol{x}}|i) P_Y(i)}{P_X(\vec{\boldsymbol{x}})} \right\} ,$$

Informally, this equation says that the posterior probability of a pixel is equal to the class likelihood $P_{X|Y}(\vec{\boldsymbol{x}}|i)$ multiplied by the prior class probability $P_Y(i)$, all divided by prior evidence of pixel $P_X(\vec{\boldsymbol{x}})$. Taking the log and dropping the denominator as it ultimately does not change the optimization process, we get:

$$i^*(\vec{\boldsymbol{x}}) = \arg\max_i \left\{ \log P_{X|Y}(\vec{\boldsymbol{x}}|i) + \log P_Y(i) \right\} . \quad (1)$$

This is the rule in which we can classify each pixel to a given class. By computing this value for each class per pixel, we can retrieve the class which maximizes this value.

As stated previously, we chose a single Gaussian model as our class distributions. In other words, under the Gaussian assumption, the probability for each pixel given a class is:

$$P_{X|Y}(\vec{\boldsymbol{x}}|i) = \frac{1}{\sqrt{(2\pi)^d |\Sigma_i|}} e^{-\frac{1}{2}(\vec{\boldsymbol{x}}-\mu_i)^\intercal \Sigma_i^{-1} (\vec{\boldsymbol{x}}-\mu_i)} . \quad (2)$$

Inserting (2) in (1), the decision rule thus becomes:

$$\begin{aligned} i^*(\vec{\boldsymbol{x}}) = \arg\max_i \Big\{ & -\frac{1}{2}(\vec{\boldsymbol{x}}-\mu_i)^T \Sigma_i^{-1} (\vec{\boldsymbol{x}}-\mu_i) \\ & -\frac{1}{2}\log(2\pi)^d |\Sigma_i| + \log P_Y(i) \Big\} , \end{aligned} \quad (3)$$

where, again, $\vec{\boldsymbol{x}}$ is the 3-dimensional vector of RGB values for a given pixel (after preprocessing), and d is the dimension of features. $P_Y(i)$ is the prior class probability, calculated from the training data by dividing the number of pixels per class by the total number of pixels.

To compute (3) (i.e., the BDR), we need to first construct the mean and covariance vectors/matrices for each class. For the mean, we can calculate:

$$\mu_i = \frac{1}{N} \sum_{n=1}^{N} x_n^i , \quad (4)$$

and for the covariance:

$$\Sigma_i = \begin{bmatrix} cov(x_i, x_i) & cov(x_i, y_i) & cov(x_i, z_i) \\ cov(y_i, x_i) & cov(y_i, y_i) & cov(y_i, z_i) \\ cov(z_i, x_i) & cov(z_i, y_i) & cov(z_i, z_i) \end{bmatrix} \quad (5)$$

where:

$$cov(x, y) = \frac{1}{N-1} \sum_{n=1}^{N} (x_n - \bar{x})(y_n - \bar{y}) ,$$

for N number of training data points.

b) Training: From the above, we can see that the training task of our machine learning model is to calculate the mean μ (4) and covariance Σ (5) for each class i using N number of training data points. To do this, prior to inference and classification, in our training procedure we hand-labeled 70 images and extracted three specific classes: "cup red", "not cup red", and "other". We used MATLAB's Image Labeler application for this task, where we segment each training image accordingly and append those three values per pixel to a *.mat* file with its corresponding label. Equations (4) and (5) were then subsequently calculated in a separate script as a class' Gaussian model parameters to be used during inference.

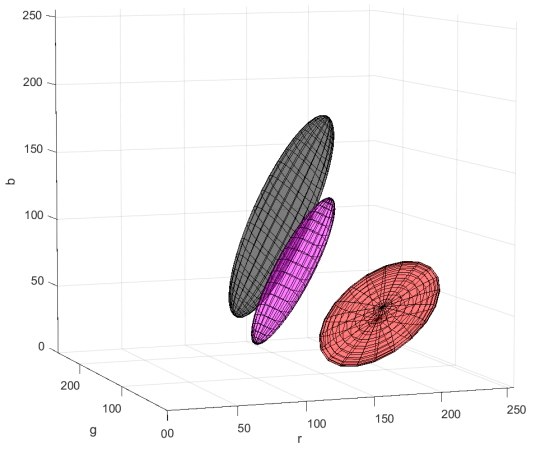


Fig. 4: **Trained Gaussian Models** A visualization of our trained single Gaussian models for our three classes, with the color representing the class. This is plotted in RGB space.

c) Inference: For pixel-wise classification during inference, we used the trained model parameters as described in the previous section to calculate the BDR value for each pixel using (3). In other words, inserting the estimated (trained) class model parameters μ and Σ into (3) for each pixel, we can select the class i which maximizes the Bayes Decision Rule equation. Thus, after doing so for each pixel, we can construct a binary mask of the original input image, in which $True$ values correspond to pixels which are thought to be the object of interest (cup), and $False$ values are pixels that do not correspond to the cup.

A brief note on computational complexity: because we need to calculate the BDR of every pixel of every frame for each class, this comes out to be 1200x900x3 = $3,240,000$ calculations *per image*. While computational complexity is still linear (i.e., $\mathcal{O}(W\text{x}L\text{x}C\text{x}N)$), in practice this is still extremely computationally inefficient and can be frustrating to deal with. To mitigate this, we vectorized these computations using Python's $einsum$ function to calculate the BDR of all $1,080,000$ pixels in a frame at once for each class. By doing so, we only needed to iterate over the number of classes (3) rather than needing to loop over all pixels and classes. The result of this is an extremely efficient method of pixel-wise classification, with one frame taking approximately one second to compute.

3) Bounding Box Detection: Once the binary mask has been extracted after classifying each pixel as "1" for the red cup and "0" for all other classes, we can now perform post-processing to extract the desired bounding box. More specifically, using the low-level pixel-wise classification, we can now extract higher-level bounding boxes to locate and detect our red cup in the image. This step is necessary as there will more than likely be erroneous pixel classification due to noise, unoptimized model parameters after training, an overly-simplistic model, or other factors. In this step, we employ simple and intuitive geometric constraints on each masks' "True" region. These regions are first filtered through a series of pre-determined heuristics, then ranked in terms of their calculated "cup-ness" and selected whether they meet these criterion or not.

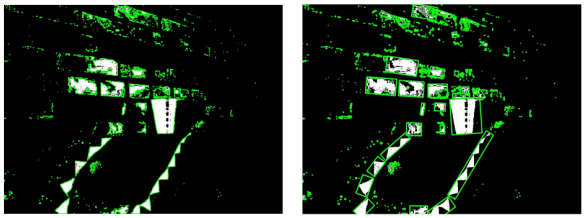


Fig. 5: **Contours and Rectangle Extraction.** Example contours (left) and rectangles (right) extracted after pixel-wise classification. The image on the right is subsequently fed into our geometric heuristics in order to detect the final bounding box (shown in Fig. 1

More specifically, from the binary mask, we first extracted contours via OpenCV's "findContours" function, which is essentially a type of edge detection on the mask. From these contours, we searched and fitted rotated rectangles — or in other words, candidate bounding boxes. These candidate bounding boxes were then filtered using a series of geometric criteria, including:

- Each side of a bounding box must be at least 10 pixels in length. This was used to filter out small bad patches and pixel-wise noise.
- The height-width ratio of a bounding box cannot be greater than four. In other words, boxes that were too elongated were filtered out, as our cup had a much smaller height-width ratio.
- The area of the bounding box must be no less than 100 pixels and no more than $1,500$ pixels.
- The magnitude of error between the height-width ratio of a bounding box and the original cup is less than some arbitrary threshold. In other words:

$$\left| \frac{h_{bb}}{w_{bb}} - \frac{12.5\text{cm}}{10\text{cm}} \right| \leq threshold\,,$$

where $threshold$ was selected to be 1.25 empirically.

The bounding boxes that were left after these series of heuristics were then final candidates of bounding boxes for our cup detection, in which the one with the largest area was selected as the final detection.

C. Deep-Learning Object Detection

1) Data Preparation: With the same original 70 images that we used to train our probabilistic pixel-wise classifier, we prepared a separate set of ground truth labels to be used for training / fine-tuning YOLOv3. While the ground truth labels were assigned pixel-wise in the pixel-wise classifier, the ground truth labels here were assigned by specifying the bounding box around the red cup. "LabelImg" was used for the labeling task, as it conveniently outputs text files containing the appropriate label format that YOLOv3 can read (i.e. the center coordinates, width and height of the bounding box). When drawing the ground truth bounding boxes, for consistency, we drew the boxes conservativley, as close to the red cup as possible, not leaving too much empty

Fig. 6: **YOLOv3 Training Loss Plot** Average loss quickly decreases within a couple hundred epochs, but we observed loss to continually decrease at a smaller scale, and stopped training around average loss of 0.05.

space between the cups and boxes. Lastly, we split the 70 labeled images into 56 for training and 14 for validation.

2) Training: To detect the bounding boxes of our object of interest, we fine-tuned a pre-trained model for YOLOv3 by using the weights that were pre-trained on Imagenet as weight initialization [1]. Instead of than training for three classes as we did in our traditional approach, we trained for just one class ("red cup"). We chose to train with the following hyperparmeters: batch size of 64, subdivisions of 16, and a SGD optimizer with learning rate of 0.001, weight decay of 0.0005 and momentum of 0.9. Training was done using an NVIDIA TITAN RTX GPU for a $3,100$ epochs. This number of epochs to train for was determined empirically, where we watched the average loss per epoch, and stopped training when it has reached around 0.05 and was fluctuating and not decreasing much more. Training took roughly around 3 hours. The loss function graph can be seen to verify sufficient training and tapering of the loss function.

3) Inference: Once the weights at $3,100$ epochs were obtained, we were able to feed our model test images of red cups for detection one by one. Upon detection, the original YOLOv3 outputs the percentage sureness of detected objects (i.e. if it detects a cup, how certain is it that it is a cup) and an image with detection boxes overlaying the original image. We modified the original code to also output, in text, information regarding the box's location and dimension.

III. Results

A. Performance Comparison

To compare the performance results between the Pixel-Estimation method and the YOLOv3 method, several comparison metrics were developed to compare our prediction boxes against the ground truth boxes developed during the data preparation phase of YOLOv3:

- Correct Detection: a correct detection is determined qualitatively by looking at the photos and deciding if the prediction box is boxing the red cup.
- 4-Corner Euclidean Distances: computes the summed up euclidean distances of four corners of the predicted detection box to the respective four corners of the ground truth box, i.e.:

$$dist = \sum_{n=1}^{4} \left\| p_i^2 - g_i^2 \right\|_2 \tag{6}$$

where p_i is the i^{th} corner coordinate of the predicted box, and g_i is the i^{th} corner coordinate of the ground truth box.

- Center Euclidean Distance: computes the euclidean distance of the center of the predicted detection box to the center of the ground truth box, i.e.:

$$dist = \left\| p_c - g_c \right\|_2 \tag{7}$$

where p_c and g_c are the center coordinates of the predicted box and ground truth box respectively.

TABLE I: Performance Comparison Under Different Metrics

	Pixel-Wise Classifier	YOLOv3
Correct Classifications	28/30	29/30
Average 4-Corner *	73.34px	51.31px
Average Center *	11.76px	4.89px

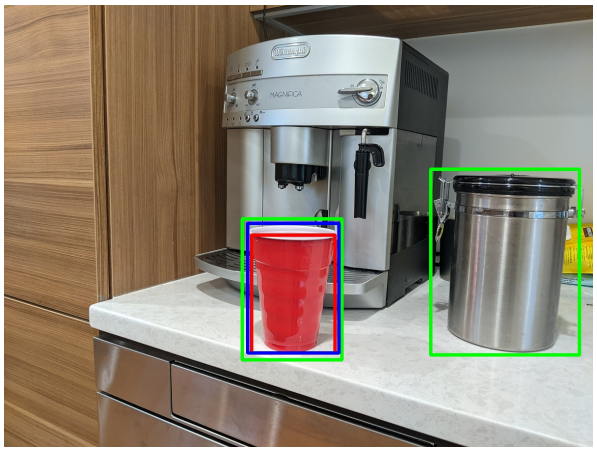
*Average distances taken across 27 out of the 30 test images that were correctly classified by both classifiers.

As seen in Table 1, our metrics reflect that the modern deep learning approach of YOLOv3 performs better than the probabilistic Pixel-Wise Classifier that we developed. However, we note that this was to be expected and that the computational expense for training was far larger in the deep learning approach than that of the traditional one.

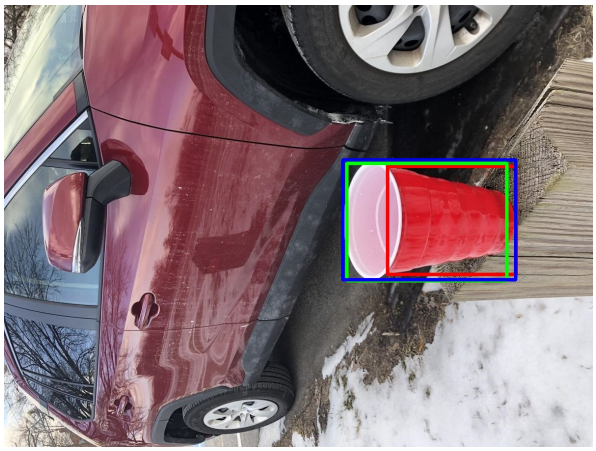
B. Qualitative Comparison

For a more visual comparison, below are examples of test images with the following overlayed on top: prediction box generated by the Pixel-Wise Classifier (in red), prediction box generated by YOLOv3 (in green), and the ground truth prediction box (in blue).

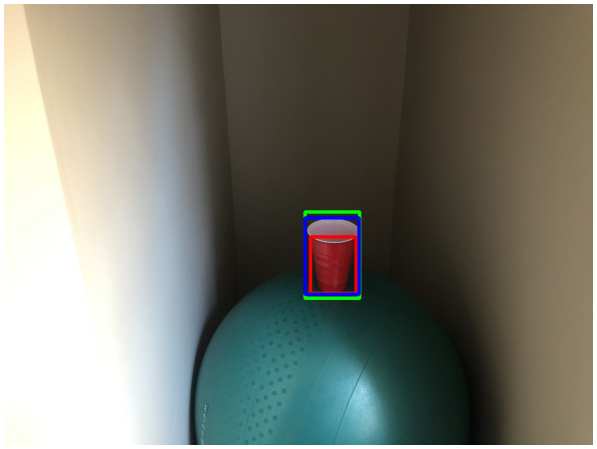
Test No. 25: Example of pixel-wise classifier (red) failing to detect the red cup, while YOLOv3 (green) correctly identified the red cup.



Test No. 30: Example of pixel-wise classifier (red) correctly identifying the red cup, while YOLOv3 (green) detected the red cup, but also mistakenly identified another item.

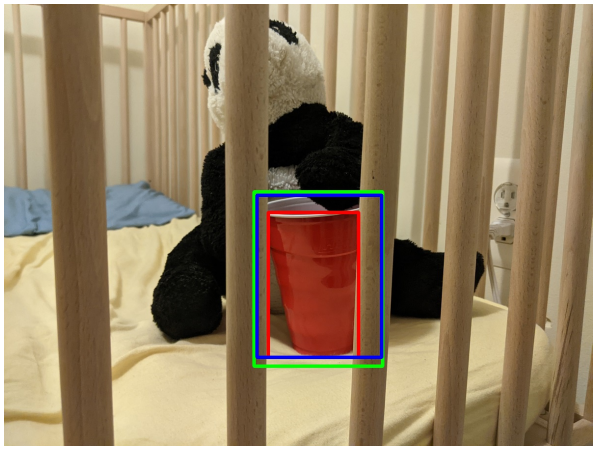


Test No. 10: Example of correct boxing using both methods. Pixel-wise method (red) 4-corner-metric: 207.67, center-metric: 46.01, and YOLOv3 (green) 56.3, 6.8 respectively.



Test No. 27: Example of correct boxing using both methods. Pixel-wise method (red) 4-corner-metric: 97.49, center-metric: 18.11, and YOLOv3 (green) 40.1, 2.5 respectively.

Test No. 0: Example of correct boxing using both methods. Pixel-wise method (red) 4-corner-metric: 47.43, center-metric: 11.18, and YOLOv3 (green) 46.9, 1.58 respectively.



Test No. 10: Example of correct boxing using both methods. Pixel-wise method (red) 4-corner-metric: 170.48, center-metric: 20.81, and YOLOv3 (green) 53.58, 2.26 respectively.

Test No. 1: Example of correct boxing using both methods. Pixel-wise method (red) 4-corner-metric: 36.77, center-metric: 8.06, and YOLOv3 (green) 10.42, 2.69 respectively.

Test No. 9: Example of correct boxing using both methods. Pixel-wise method (red) 4-corner-metric: 20.0, center-metric: 2.24, and YOLOv3 (green) 35.62, 2.55 respectively.

IV. Discussion

In this project, we developed and compared two separate pipelines for object detection of a red cup, one using traditional probabilistic learning and the other using modern neural networks. In our statistical pipeline, we first compensated for lighting invariances across our novel dataset by histogram-equalizing the value (brightness) channel of each HSV-converted image. Afterwards, we implemented a single Gaussian pixel-wise classifier model to extract the pixels which were most probable to be part of the cup using trained model parameters. The output mask was then subsequently fed through a series of geometric heuristics to ultimately output our desired bounding box. On the other hand, in our deep learning pipeline, we used a fully convolutional neural network architecture via YOLOv3 to compress an image into a latent space representation, which outputted the bounding box of our desired object through a fine-tuned model.

Through this project, we gained several important insights into the difference between traditional, statistical learning approaches that recognize patterns in the training set, and more modern deep-learning approaches using neural networks (which are much more opaque than the classical methods). First, we note that modern vision methods using neural networks are by far more robust; this is because convolutional neural networks look at images through several a sliding filters, which allows the output latent representation to hold the most valuable information of the object. In other words, these networks look at objects more wholistically. In contrast, our pixel-wise method, while it does employ geometric constraints during post-processing, classifies per-pixel and is much more susceptible to errorneous classifications. However, we note that these traditional methods are much more transparent in that every step is clear by using first principles. This can provide benefit to those who wish to further improve such pipelines by creating an architecture which can be easily modified. On the other hand, neural networks are much more of a "black-box" and opaque and modification/improvement is using done at a much higher level; however, such structures are experimentally much more robust and are far more accurate in practice, as indicated by the advent of such techniques in recent years.

By implementing these two approaches for the same computer vision task, we compared their performances in accuracy and computing time, practiced different machine learning techniques taught in this course, and familiarized ourselves with image data processing, computer vision techniques, and the OpenCV [9] library. In the end, we hope we have provided the class with a deeper understanding of how each technique works at a fundamental level, and why the transition to modern approaches was necessary in improving performance for object detection.

References

[1] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *arXiv preprint arXiv:1804.02767*, 2018.

[2] Z. Zou, Z. Shi, Y. Guo, and J. Ye, "Object detection in 20 years: A survey," *arXiv preprint arXiv:1905.05055*, 2019.

[3] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *Proceedings of the 2001 IEEE computer society conference on computer vision and pattern recognition. CVPR 2001*, vol. 1. IEEE, 2001, pp. I–I.

[4] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*, vol. 1. Ieee, 2005, pp. 886–893.

[5] P. Felzenszwalb, D. McAllester, and D. Ramanan, "A discriminatively trained, multiscale, deformable part model," in *2008 IEEE conference on computer vision and pattern recognition*. IEEE, 2008, pp. 1–8.

[6] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial pyramid pooling in deep convolutional networks for visual recognition," *IEEE transactions on pattern analysis and machine intelligence*, vol. 37, no. 9, pp. 1904–1916, 2015.

[7] R. Girshick, "Fast r-cnn," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1440–1448.

[8] N. J. Sanket. Cmsc426 computer vision. [Online]. Available: https://cmsc426.github.io/colorseg/

[9] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.