ECSE 415 Project Report

McGill University, Fall 2020

Group #: *12*

Group Members:

- Fei Peng 260712440

- Yukai Zhang 260710915

- Yinuo Wang 260766084

- Lawrence Zhang 260664442

Description of Person Detector (part 1):

Describe the method used to detect people in part 1. State from where the method was obtained (e.g. website or GitHub repository) and briefly describe how the method works. Provide an image showing the detections obtained on at least one image from the dataset.

(a) (b)

Figure 1: Object detection from Facebook Detectron2 (a), augmented people detection (b).

The method we used to detect people in part 1 is originally from Facebook Detectron2 (https://github.com/facebookresearch/detectron2) [5] and augmented to suit our use. Figure 1 (a) shows the object detection from Detectron2, whereas Figure 1 (b) illustrates the people detection of the augmented version. Detectron2 is powered by PyTorch (i.e., a deep learning framework), and includes models such as Faster R-CNN, Mask R-CNN, RetinaNet, DensePose, Cascade R-CNN, Panoptic FPN, and TensorMask. We selected a pre-trained Mask-RCNN model and its config file (e.g.., its weights) from model zoo, namely mask_rcnn_R_50_FPN_3x.yaml, and used it to perform object detection. Mask-RCNN is a deep neural

network aimed to solve instance segmentation problems in machine learning or computer vision, which can separate different objects in an image. There are two stages of Mask-RCNN (i.e., first, object regions proposal, second, class prediction, bounding box refinement, and box generation), which are both connected to the backbone structure (e.g., FPN style deep neural network) [5, 11]. Since it is a multi-class detection method, we tweaked it and selected people to class only (i.e., class 0). To do that, we extracted the bounding boxes of class 0 alone, around which we drew rectangles. Therefore, only detections concerning people would be shown and counted towards the total number of people in the mall. According to our observation, our augmented Detectron2 had an excellent and outstanding behavior on people detection, and there was no duplicated detection, thus, we did not apply any additional duplication removal in this part. However, non-maximum suppression (i.e., NMS) was performed in part 2. We evaluated the person counting method using Kaggle and obtained a weighted mean absolute error of 0.10540. Details of NMS and the performance of augmented Detectron2 will be discussed later in the following sections.

Description of Person Detector (part 2):

Describe the method used to detect people in part 2. Briefly describe how the method works. Describe what features you used to feed into the SVM classifier. Explain your choice of window size. Outline the process used to get the training data (positive and negative examples). Explain your choice of the number of training examples.

The methods we used to detect people in part 2 consist of three parts: extracting positive and negative sample patches, applying features descriptor (i.e., HoG [1, 8] from OpenCV, Haar [2, 10] and LBP [3, 10] from scikit-image), and training the SVM classifier (SVC from scikit-learn) [4, 9].

In the feature extraction part, we first ran the Detectron2 algorithm as described in part 1 to identify people. Detected people would be surrounded by red boxes as shown in Figure 2 (a), and their coordinates would be recorded. By extracting the pixels within red boxes from the images, we could obtain positive sample patches. We also needed negative sample patches to complete the training set, since this is a two-class classification problem (i.e., people and non-people). Thus, we drew boxes of different sizes in the image and eliminated those that were intersecting with any of the positive boxes previously drawn by the Detectron2 algorithm. The negative patches are shown in Figure 2 (b)).

The size of the training set can greatly affect the performance of SVM training. An oversized training set may result in overfitting, however, insufficient training data (i.e., fat matrix) may lower the accuracy. Moreover, we would need both training and validation data to evaluate the performance of the SVM classifier.

Therefore, after careful consideration, for HoG and LBP, we decided to extract positive patches from 200 random images and negative patches from 200 different random images as our dataset, but for Haar, that number would be 100 images. When the Detectron2 algorithm was applied, an average of 24 people could be detected from each image so 200 images would provide us approximately 4800 positive sample patches (2400 for Haar). To have a balanced dataset, we expected the number of negative sample patches to be close to that of the positive ones. However, there are usually more negative patches available in an image than positive patches. Thus, for each image, we first counted the number of positive patches it contains and then selected the same number of negative patches.

(a) (b)

Figure 2: Positive sample patches (a) and Negative sample patches (b) of people.

After obtaining both the positive and negative sample patches of the images, we changed them all to the same size (i.e., cv2.resize()) and expressed them using different feature descriptors. We tried the histogram of oriented gradients (HoG) [1], Haar feature [2] as well as Local Binary Pattern (LBP) [3]. The result of feature descriptors are concatenated and the labels (0 for negative and 1 for positive) are appended to form the input data and labels. They were later used as the training/validation dataset of the support vector machines (SVM).

Our implementation of **HoG feature descriptor** first divides the image window into multiple 1 by 3 pixels cells. For the pixels in each cell, its gradient direction/edge orientations are calculated to form a histogram which will be later combined and used to represent the feature. Multiple blocks each consisting of 4 by 4 cells are formed such that a contrast normalization can be performed and the feature will be less likely affected by the illumination and shadowing [1].

After training the SVM classifier with the feature obtained by HoG, we implemented a sliding window and a pyramid function to perform people detection. The sliding window function extracts image patches of size 16 pixels wide and 48 pixels high. The width-height ratio was chosen to be 1:3 such that it matched the majority patch sizes obtained in part 1. The pyramid function scales down the original image by a factor of 2/3 per loop (i.e., previous image size equals 1.5 times the next image size) such that the sliding window function can cover different sizes of regions. As the original image shrinks, the size of each patch obtained from the sliding window gets relatively larger. We limited the minimal size of the shrunk image to avoid having oversized windows and wasting computational power. This helped us obtain an additional two sizes of windows: 24 by 72 pixels and 36 by 108 pixels. Extracted windows were resized before calculating their HoG features. The SVM classifier we trained will then use these features to determine if the window contains a person. After the pyramid sliding window was complete, we applied the non-maximum suppression algorithm to discard the duplicated patches before we counted the total number of people in an image.

Another feature we used to train SVM in the second part of this project is **the Haar-like feature**. It mainly uses a collection of small rectangular kernels to extract certain significant patterns on an integral image. Haar-like features encode the selected region of an image into a vector which could be used as a detailed

descriptor. Also, generated from integral images, the calculation is also straightforward (mostly additions) [2]. Considering the above characteristics of Haar features, it is reasonable to assume that Haar-like features could serve as a robust descriptor for training our SVM.

Starting with employing the *haar_like_feature()* function from sklearn library, we conveniently extracted Haar features from patches containing people and non-people patterns generated from part 1. We chose 100 images as the data pool, containing over 2000 different patches for positive and negative patches respectively. Each patch is resized to a slim vertical rectangular region of 5×12 pixels. What follows the resizing is the extraction of type-2-y features. Those parameter decisions are decided upon trials and they represent the final compromise between accuracy and speed (more explanation in later sections). Then, those features are normalized, shuffled, and split into training and testing with a ratio of 80% training and 20% testing, the same as the LBP case. Next, a linear SVM with auto-chosen gamma value is created and trained, and later achieves an accuracy rate of over 80% on average. Finally, the trained SVM is used in the sliding window algorithm described previously. Note we set a larger window size (27×81) and smaller step size (10) for Haar features. For more explanation please refer to the later analysis on performance.

Local Binary Patterns is a method that compares a central pixel to neighboring pixels, determines whether or not they are larger or smaller in value (usually in grayscale), and assigns the central pixel with a binary number based on the comparison with its neighboring pixels. By doing this relative calculation, it can determine the edges, flat surfaces, and corners of an image. The features can be computed with these capabilities in mind. Our implementation was to first define the function that calculates the features by setting the radius to 1 and the number of relative points to 8. We then proceed to resize the areas that the detectron thinks contains people and compute the local binary patterns of these resized croppings. Using the function we defined, we compute the positive and negative LBP features from a random sample size of 200 each. After concatenating these, we can fit it on the SVC. The reason we chose 200 was that anything below that would affect our accuracy, but anything above that would make Google Colab run that block of code forever without returning results until it times out. Choosing our window size to be (72,24) was because this pixel size is enough to outline the shape of a person walking in a mall.

To find the best feature descriptor, 80% of the data was used to train the SVM and the remaining 20% was used as the validation set. The validation accuracy of HoG, Haar, and LBP was 87.92%, 83.58%, and 82.29% respectively. HoG was found to have the highest validation. Haar and LBP, on the other hand, both yielded slightly lower validation accuracies. To decide which feature descriptor is more suitable for this project, we still need to evaluate the result using the IoU metrics.

Description of the duplicate detection removal and person counting:

Describe the method used to remove duplicate detections in part 2. Briefly describe how the method works. Also, describe the code used to count the detections.

When applying the sliding window and pyramid function, duplicated detections will likely be made on the same person. Thus, a non-max suppression function is called every time we finish scanning an image.

This function will check the ratio of intersection area over union area for the overlapping patches. If for any overlapping patches, this ratio is higher than 0.1, only the one that better represents the people (i.e., with

higher score) will be kept. We always keep the patch that is further away from the SVM boundary line such that they will less likely be misclassified.

The coordinates of the boxes that contain people after applying the non-max suppression will be returned in a list. Each element in that list is the coordinates of the boundary box containing the detected people and the length of the returned list represents the number of people in the image.

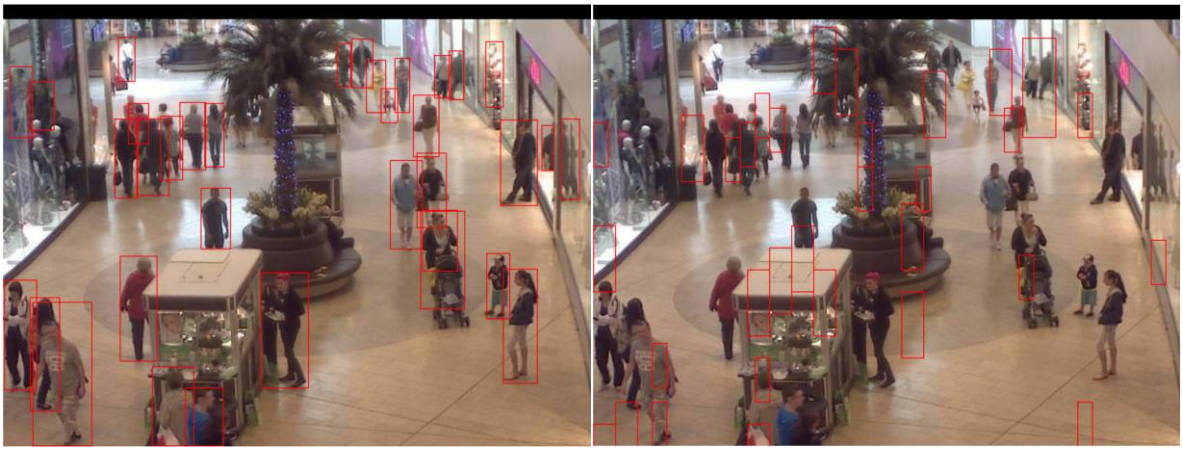
Evaluation of Person Detector:

Describe how you evaluated the performance of your person detector. Take the ground truth to be the bounding boxes detected in part 1. Use the IoU metric to quantify the performance of the detector. Provide an image showing the detections obtained on at least one image from the dataset.

The performance of the person detector is evaluated using the intersection over union (i.e., IoU) metric, which is the most popular evaluation metric used in the object detection benchmarks [7]. An example of the ground truth of the bounding boxes obtained from part 1 is shown in Figure 3 (a) and Figure 4 (a) and the results of our HoG-featured, LBP-featured, and Haar-featured SVM person detector are shown in Figure 3 (b), Figure 4 (b) and Figure 6 (b), respectively. To perform the IoU evaluation, we created two 480 by 640 pixel mask matrices. If a pixel in an image is within any of the ground truth boxes, its corresponding value in the mask matrices is set to 1, otherwise, it is set to 0. A likewise operation is performed for the SVM detection result. After computing the sum of the two masks, there will be three possible values in the SVM detector. 2 stands for the intersection area and the combination of 1 and 2 stands for the union area. By dividing the number of 2s by the number of 1s plus 2s, we can obtain the IoU. Figure 3 illustrates a randomly selected image with HoG-featured SVM detection, which obtained an IoU of 0.42012. Figure 4 is an example of an **LBP-featured** detection result with an IoU of 0.27943.

(a) (b)

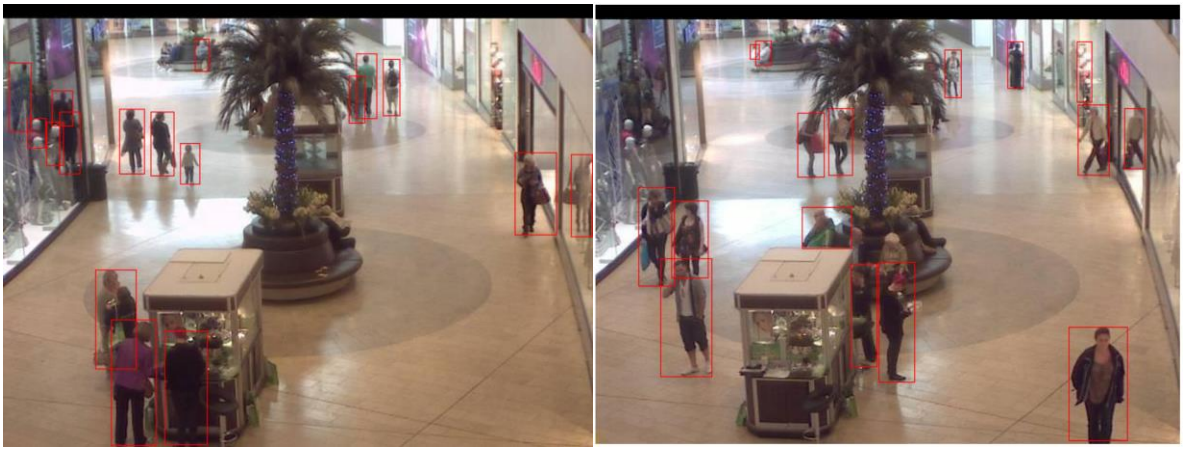
Figure 3: Ground truth (a) and the result of SVM person detection (HoG) (b).



(a) (b)

Figure 4: Ground truth (a) and the result of SVM person detection (LBP) (b).

Note that in Figure 3 (b), several boxes are surrounding the mannequins which are behind the glass window on the top left corner. They were treated as misclassified because some of the training samples obtained in part 1 treat the mannequins as people and some do not. Examples are shown in Figure 5. If the Detectron2 algorithm could generate a more consistent people detection result, the SVM classifier would obtain a better IoU.



(a) (b)

Figure 5: Examples of misclassified ground truth obtained from part 1.

By using **Haar feature descriptor**, the result we get over 2000 frames is an average IoU of 0.261409. The best attempt over these trials is an IoU of 0.42012. Sources of errors include capturing only a part of the pedestrian, capturing other pixel patterns that look like people (the garbage bin in Figure 6 below for example), and incorrect merges of regions with no people. A comparison between the ground truth and the SVM test is shown below.

(a) (b)

Figure 6: Ground truth (a) and the result of SVM person detection (Haar) (b).

Evaluation of Person Counting:

Run the person detector of part 2 on all 1000 images of the dataset. Construct the response spreadsheet and upload it to the Kaggle competition website. List the score (ranking and metric) reported on the Kaggle website leaderboard. (note: this will not be the final score/ranking, as that will only be reported after the end of the competition period).

The person detector developed in part 2 was used for the people detection of all 2000 images provided and the result was uploaded to the Kaggle competition website. The weighted mean absolute error of our SVM classifier is found to be 0.17565 (HoG Feature Descriptor), 0.15248 (Haar Feature Descriptor), 0.21026 (LBP Feature Descriptor), and that of Detectron2 is 0.10540, ranked the 3rd. However, near the end of this project, we found in the MyCourses Discussion Board that Prof. James Clark stated *"it seems that in looking at the ground truth counts that the mannequins are counted, as well as the reflections of people in the store windows."*, which was not an official announcement yet. Therefore, we slightly adjusted our submission file (for Kaggle) to adapt to this change, by adding a constant offset value to all the counting results.

Discussion:

Discuss the lessons learned during this project. How could you improve upon your results? Are there better ways to estimate how many people are in an image than the approach used in this project? Describe any difficulties you faced in the project. Discuss the suitability of different types of features (e.g. HoG vs Haar or LBP or even Deep Features provided by a CNN).

After developing the person detection algorithm, we found that the best feature descriptor suitable for this project is Haar. By comparing the number of duplicated detection before and after applying a non-max suppression, we also learned that NMS can greatly improve the accuracy of counting the number of people in a mall. IoU was found to be a fair metric to evaluate our algorithm since it can directly tell us how much discrepancy is there between our people detection and the ground truth.

To improve upon our result, we should first take a closer look at our ground truth/training sample. The training sample patches are obtained by applying Detectron2 algorithm provided by Facebook on the

images. Although they have a rather good person detection, it is found that some but not all of the detections concentrate on the mirror image on the glass wall and the mannequins, these misclassified patches could be eliminated by applying the region of interest (i.e., ROI) limitation so that the training sample would be more accurate. For negative samples, we can manually choose some negative image patches based on the misclassified boxes we obtained in the SVM classifier so that the chance of misclassification will be lower.

We should also reduce the step size when performing the sliding window and reduce the shrinking ratio when performing a pyramid function. This will help us to have a more thorough scan and it would be easier for us to choose the best boundary box containing a person after the non-max suppression.

Deep learning is one of the many alternative approaches to estimate the number of people in an image and it might yield a better result. However, due to the limitation of our computational power and dataset, a fully developed deep-learning-based classifier may not be properly trained. Thus, it may be our future development goal.

One of the many difficulties we encountered is the limitation of computational resources. To train the SVM, it is essential to obtain the correct sample patches from part 1. Thus, we needed to use Detectron2 to perform person detection for all 2000 images every time before we needed to make alterations on the SVM model. The sample extraction process requires a great deal of computational resources and with the GPU provided in Google Colab, this process usually takes over an hour. It greatly slows down the progress of our development. Thus, we converted all the data required for the future SVM training into multiple data packages (e.g., positive patches, negative patches, and trained SVM model) so that it can be easily dumped or loaded. Loading data takes approximately 5 seconds which greatly boosts our development efficiency.

The main issue we discovered with **HoG and LBP features** was that on almost every example frame, the program would only have one out of 20 sliding windows correctly positioned on a person. As we mentioned previously, we could not increase the sample size for training since the free amount of memory allocated by Google Colab was very limited. We also couldn't decrease step size since that would increase the execution time multiplicatively by 2000 (since we have 2000 images to process). For the HoG feature descriptor, despite having the highest validation accuracy for the SVM model, it yields a lower IoU in people detection compared to Haar. Thus, with the results we obtained, it was decided that we would not proceed to train with HoG or LBP features.

We undergo three stages when attempting to implement **the Haar feature** for SVM. In this subsection, we will discuss the lesson we learned and the advantage or drawbacks of Haar-featured SVM.

During the first stage, what we did was set up the program and tried to see what could happen with the SVM. For the first few tests, we witnessed a very slow running speed of SVM training. Even training 1 image (about 20 patches) with a linear SVM takes forever. The accuracy of the endless training is also not satisfying, only at around 50%. Based on the observation later, we concluded that the huge size of the Haar vector and the numerically unstable value in the Haar vector is what caused the initial SVM attempt to suffer. We initially extract Haar features from an unnormalized region of 8×24, which returns a vector of approximately 8000 features. Within each vector, some values, both negative and positive, can have an absolute value up to 10^{24}. Thus, we add scaling (i.e., normalization) blocks to the train and testing vectors. Also, we lowered the size of the interested region down to 5×12 (almost ¼ of the area). Now, every Haar

vector contains only several hundred features. We successfully implemented a functional SVM, acting in a timely manner.

During the second stage, we tried getting different types of Haar features. Initially, we asked for both features in the x and y direction by setting the feature parameter with “type-2-x, type-2-y”. However, after evaluating the positive (people) patches, we found that most samples are slim vertical rectangles. Thus, extracting x features might not be as useful, considering the patches have been resized into a 5×12 integral image with a lot of the information on the x-axis compressed and distorted. In the final solution, we decided to take only y-features. This decision increased speed because it further shortened the Haar vector to a size of 462. Also, since most of the people patches are standing or walking figures, vertical filters capture the significant features. The accuracy of SVM using only vertical features is boosted to 80%, with the best trial of 86%.

The last stage is using the trained SVM and testing it out on the frame. Despite that Haar-featured SVM generates some false positives results, its IoU is still acceptable. We tried enlarging the window size so that the entire people can be recognized and captured (but also creating internal fragmentations which lower IoU). Also, we decreased the step size so that the windows are less likely to divide people into parts. The current parameter, win_size of (27×90) and step_size of 10 illustrated the best trade-off point, achieving an average IoU of 25%.

Based on this result, we conclude that Haar features and SVM have a fair enough synergy. However, besides factors like imperfect ground truth, we found parameters from Haar features will interfere with the performance of SVM, thus lower the performance. An example is that to get good Haar extraction, we want to resize a region as large as possible. However, as mentioned if the patch is resized to 8×24, a vector with over 8000 elements will be returned. This makes the dimension of SVM too high for a linear kernel to train. Also, numerical stability becomes questionable since there might be an overflow. On the other hand, what we did is we compressed the patch down to 5×12. Then, the new problem is that the diagram becomes very blurred, as shown in Figure 7. As the example shows below, the middle region is the resized 5×12 patch. We can barely say that it is the shape of a person. Thus, while making the Haar features computable, we sacrificed its accuracy.

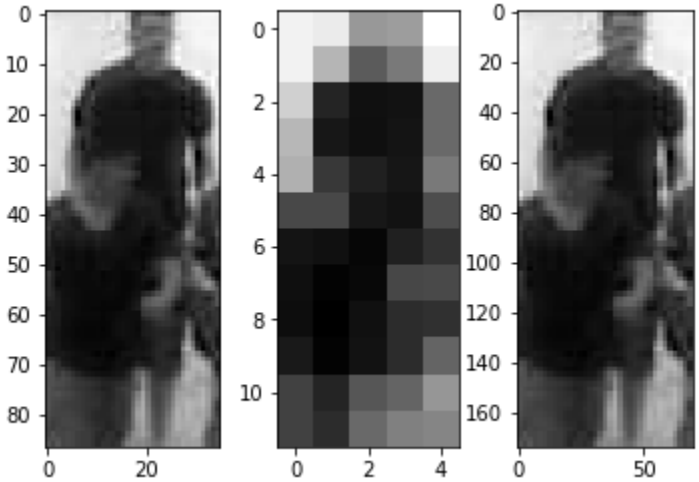


Figure 7: An comparison of the original patch (left) and resized patches used for Haar feature (middle)

Thus, we conclude that using Haar features with SVM still has a lot of improvement to make in the future. Haar features are computationally expensive. On top of that, Haar features bring too many dimensions to SVM. We believe Haar features should be used with cascade classifiers instead of a single monolithic SVM. At each stage of the classifier, non-people windows can be eliminated so that we do not need the expensive Haar extraction at every window. This would find a new and better balance point between computation speed and accuracy. The last thing worth mentioning is that for some trials Haar will generate better performance than LBP and HoG. However, it does not prove the absolute superiority of Haar features over the other two. This is due to the different environments we used running and testing those three features. The excessive amount of Haar feature calculation overloads the kernel on Google Colab, thus most of the training is done at anaconda localhost. Also, the large window size increases the possibility for a false detection to intersect with some parts of the ground truth, boosting up the performance for Haar feature detection to some extent.

[1] 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)*, 2005, vol. 1, pp. 886–893 vol.1.

[2] P. Viola and M. J. Jones, "Robust Real-Time Face Detection," *International Journal of Computer Vision*, vol. 57, no. 2, pp. 137-154, 2004/05/01 2004, doi: 10.1023/B:VISI.0000013087.49260.fb.

[3] Timo Ahonen, Abdenour Hadid, and Matti Pietikäinen, “Face recognition with local binary patterns,” in *In Proc. of 9th Euro15 We*, pp. 469–481.

[4] John C. Platt, “Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods,” in *ADVANCES IN LARGE MARGIN CLASSIFIERS*. 1999, pp. 61–74, MIT Press.

[5] Yuxin Wu, Alexander Kirillov, Francisco Massa, Wan-Yen Lo, and Ross Girshick, “Detectron2,” https://github.com/facebookresearch/detectron2, 2019.

[6] C. C. Loy, K. Chen, S. Gong, and T. Xiang, "Crowd Counting and Profiling: Methodology and Evaluation," in *Modeling, Simulation and Visual Analysis of Crowds: A Multidisciplinary Perspective*, S. Ali, K. Nishino, D. Manocha, and M. Shah Eds. New York, NY: Springer New York, 2013, pp. 347-382.

[7] Hamid Rezatofighi, Nathan Tsoi, JunYoung Gwak, Amir Sadeghian, Ian Reid, and Silvio Savarese, “Generalized intersection over union: A metric and a loss for bounding box regression,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[8] G. Bradski, “The OpenCV Library,” *Dr. Dobb’s Journal of Software Tools*, 2000.

[9] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, “Scikit-learn: Machine learning in Python,” *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[10] Stéfan van der Walt, Johannes L. Schönberger, Juan Nunez-Iglesias, François Boulogne, Joshua D. Warner, Neil Yager, Emmanuelle Gouillart, Tony Yu, and the scikit-image contributors, “scikit-image: image processing in Python,” *PeerJ*, vol. 2, pp. e453, 6 2014.

[11] Yang Yu, Kailiang Zhang, Li Yang, and Dongxing Zhang, “Fruit detection for strawberry harvesting robot in non-structural environment based on mask-rcnn,” *Computers and Electronics in Agriculture*, vol. 163, pp. 104846, 2019.