Waste Detection Using Different Deep Learning Methods

Gabriela Bravo-Illanes
gbravo@stanford.edu

Brynne Hurst
brynnemh@stanford.edu

Tanran Zheng
trzheng@stanford.edu

I. Introduction

Litter negatively affects our environment. Toxic material and chemicals pollute the surrounding soil, water, and air. Litter can attract pests that carry diseases and harm animal life due to ingestion, entanglement, or destruction of habitat. Current solutions depend on local authorities taking action to clean cities, officials enforcing laws that prohibit illegal dumping and littering, or communities manually removing litter. Using a machine learning (ML) approach to detect litter could help towards this effort. ML could allow authorities to keep public spaces clean using an automated monitoring system [1], [2], detecting illegal littering [3], and building automated robots for litter recollection [4], [5].

One of the challenges of detecting litter is the diversity of the background (e.g., city, beach, grassland, etc). Compared with detecting other objects, trash is more difficult to detect since it can be damaged or deformed, can vary in size, or can be occluded and camouflaged [6] [7]. The goal of this project is to explore how best to apply ML algorithms to detect recyclable objects and litter in different conditions and environments. Our approach consists of first training a SVM to determine which category schema we will adopt, and then training deep learning models (YOLO and Mask R-CNN) to detect and classify the objects in the images.

To determine the category schema we trained a support vector machine (SVM) with cropped images taken from the original data set. Comparing the confusion matrices of the model trained with different schemas, we were able to identify a schema with both low false negative and low false positive that could also give us enough information to classify between recyclable objects and litter.

For detecting the objects, we chose to implement two popular and effective models: YOLO and Mask R-CNN. Both are also the most representative models of two mainstream object detection algorithms: region proposal based and regression based, respectively. With these two models, we were able to predict both the positions and categories of the objects from raw images (not cropped).

II. Related Work

The problem of image classification applied to recyclable objects was previously covered by two CS 229 teams with the goal of improving sorting of recyclable objects. In 2016, Mindy Yang and Gary Thung [8] collected and classified images with a single object such as glass, paper, metal, plastic, and cardboard with around 400 to 500 images for each class. They trained a SVM with scale-invariant feature transform (SIFT) features and a convolutional neural network (CNN). The SVM achieved a test accuracy of 63% and the CNN achieved 22% test accuracy. They also made their dataset available as the TrashNet dataset [9]. Later, in 2017, Oluwasanya Awe, Robel Mengistu and Vikram Sreedhar [10] used the TrashNet data set, and artificially augmented it by overlapping the objects to create images with multiple objects. They classified the objects as landfill, recycling, and paper. They used a Faster R-CNN and obtained a mean average precision (mAP) of 68.3%. In contrast with our work, both projects used images with a solid background (the focus of the image being the object), and trained architectures simpler than YOLO.

Other authors have focused on detecting and localizing waste objects in different environments. Fulton et al. [4] classified images as plastic, man-made objects, and biological material in underwater environments. They compared the performance of four network architectures: YOLOv2, Tiny-YOLO, Faster R-CNN and Single Shot MultiBox Detector (SSD), where Faster R-CNN was the most accurate with an mAP of 81.0%. Ping et al. [1] worked on classification of street litter in 11 classes using Faster R-CNN and edge computing. They obtained a mAP@50 of $\sim$43%. Finally, Proença et al. [6] used Mask R-CNN to classify ten types of litter using the TACO dataset reaching AP class score of 17.6 ± 1.6. Although Proença used the same dataset as this paper, our work differs from the work of Proença in the model used and the class schema (they included background as one of their categories). For image augmentation, we both used image augmentation techniques from the `imgaug` Python library [11].

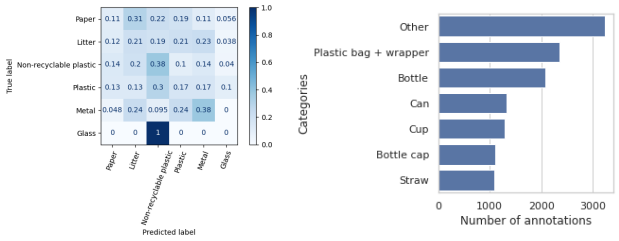
III. Dataset

The dataset we are using for this project is TACO [6], which is an open image dataset for litter detection and segmentation. It contains 1500 high resolution images of different sizes, taken by mobiles phones, showing waste in the wild. One or more objects are present in each image, having a total of 4784 annotations. The annotations are in "Common objects format" (COCO), a format that labels objects using per-instance segmentation.

A. Dataset Augmentation

In our first iteration we mapped the original 60 categories of the TACO dataset to six categories representing different types

of waste: *paper*, *plastic*, *non-recyclable plastic*, *metal*, *glass*, and *litter*. After getting results with a high number of false positive and false negatives on the initial SVM training (as can be seen in Fig. 1a), we decided to modify the categories and classify the images according to more distinguishable characteristics (e.g., shape) in order to improve the algorithm performance (Fig. 1b). We also removed annotations of objects smaller than 4% of the image. We then created the *other* category which is a grouping of categories with small number of annotations (< 100).



(a) Original 9 categories. (b) Final Categories

Fig. 1: Relabeling of the TACO dataset

To augment the dataset we used the `imgaug` library [11] to create new images using additive Gaussian noise, Gaussian blur, pixel dropout, and pixel addition. We also scaled, rotated, and translated the images. The images containing elements in categories with fewer annotations were augmented more times than other pictures, in order to create a more even dataset. However, this wasn't entirely possible because some images contained objects from different categories. After augmentation, the final data set contained 5418 images and 12500 annotations.

Finally the data was split into training, testing, and validation sets with a 80:10:10 ratio.

B. SVM Dataset and Features

One limitation of using a SVM to classify the TACO dataset is that SVMs expect one label per image. However, many images in the TACO dataset contain multiple objects. To resolve this issue, we cropped the images to isolate the individual objects. We then used the following feature extraction methods:

- Histogram of oriented gradients (HOG).
- Concatenation of the Hu Moment, Haralick texture feature vector, and the color histogram of the image (Haralick).
- VGG-19 convolutional neural network (VGG-19).

The HOG feature descriptor uses gradients of an image as the features. The magnitude of the gradient is large around object edges and corners [12], which is useful in object detection because the edges of an object can provide lots of insight into the identity of an object. To calculate the HOG, we first convert the image to gray scale, we then compute the HOG, and finally we scale the result to have zero mean and unit variance. An example of the HOG transformation can be seen in Fig 2.

(a) Original (b) HOG

Fig. 2: Original TACO image vs HOG image

The next feature descriptor for the SVM was a concatenation of the Hu Moment, the Haralick texture feature vector, and the color histogram of the image. The Hu Moment is a set of seven numbers (moments) proven to be invariant to image transformations like rotations and translations [13], it is useful for comparing shapes because objects of similar shape will have similar Hu Moments [14]. The Haralick texture features summarize the relative frequency of gray tones in an image [15]. The color histogram extracts histograms for each of the RGB components of the image [16]. After extracting these three values from an image we concatenate the results to reduce the correlation between images [17]. We chose to use these features because shape, texture, and color are features that humans use when categorizing an image. From this point on, we will refer to this feature descriptor as the Haralick feature.

The final feature descriptor for the SVM made use of the VGG-19 network to extract learned features from the images. VGG-19 is a 19 layer deep CNN originally developed by Karen Simonyan and Andrew Zisserman of Oxford University [18]. We specifically used the `Keras` VGG-19 model that was trained on the ImageNet dataset [19], [20]. To extract the learned features, we first resize each image to 224x224 pixels, we then run the VGG-19 predictions on the resized image, and finally we extract the features from the second fully-connected layer of the model and flatten them.

IV. Methods

The problem of detecting and differentiating recyclable objects and litter in different environments corresponds to an object detection problem, namely locating objects in an image and classifying them into preset categories.

Traditionally, classification is done using Support Vector Machines or Logistic Regression. Nowadays the most widely used algorithms for image classification are based on CNNs. They differ on network architecture, training strategy, and optimization function [21]. For object detection, CNN based models are also one of the most popular and powerful tools. They not only classify the objects, but also locate them. Examples are R-CNN, Fast R-CNN, Faster R-CNN and YOLO.

For this report we examined approaches for both image classification and object detection problems: SVM, Mask R-

CNN and YOLOv4. We used the SVM as a baseline for the YOLOv4 and Mask R-CNN.

A. SVM

SVM is a supervised ML algorithm capable of binary classification, but it can be extended to classify multiple categories by performing a "one versus all" scheme. In a "one versus all" scheme, a binary classifier is learned for each of the K categories against the other $K-1$ categories [22].

We used a SVM Classifier with stochastic gradient descent (SGD) from the `scikit` Python library. The goal of this SVM is to learn a linear scoring function, $f(x) = w^T x + b$ where $w \in \mathbb{R}^m$ are the model parameters, and $b \in \mathbb{R}$ is the intercept. To find w and b, we minimize the training error

$$E(w,b) = \frac{1}{n}\sum_{i=1}^{n} L(y_i, f(x_i)) + \alpha R(w) \tag{1}$$

where L is the hinge loss function given by

$$L(y_i, f(x_i)) = \max(0, 1 - y_i f(x_i)), \tag{2}$$

R is a regularization term, and α is the regularization strength. (This α is not to be confused with the SGD learning rate, which `scikit` computes as $1.0/(\alpha * (t + t0))$ where $t0$ is chosen based on a heuristic proposed by Leon Bottou.) We found the `scikit` default of $\alpha = 0.0001$ to be satisfactory for our training purposes. See [22] for an in depth discussion of the SGD Classifier model.

B. Mask R-CNN

Mask R-CNN is a region proposal based object detection algorithm, which consists of two steps: (i) compute proposed object bounding boxes and (ii) propose a classification, as shown in Fig. 3. More specifically, in the first step, the model extracts features (backbone) and generates a Region of Interest (RoI) for each image. Each RoI will be assigned a score which indicates the probability of having an object inside the RoI. In the second stage, the model predicts both the location and category of the object, by predicting bounding boxes and the object class for each RoI [23] [21]. During training, the model is trying to minimize the loss for each RoI, which is defined as:

$$L = L_{cls} + L_{boxes} + L_{mask} \tag{3}$$

where L_{cls} represents the classification loss, L_{boxes} represents the bounding-box loss, and L_{mask} represents the average binary cross-entropy loss of the mask. For more details of the model, please refer to [23].

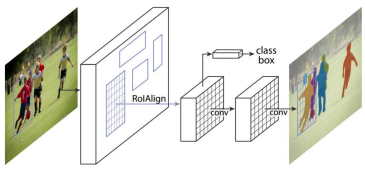


Fig. 3: Mask R-CNN Framework [23]

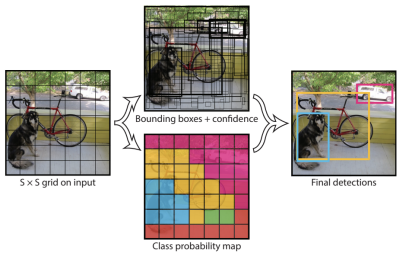


Fig. 4: YOLO Framework [24]

C. YOLO

YOLO is a regression based framework. It is a great method for this project due to its accuracy and speed on real-time prediction. The latter characteristic is critical for waste detection applications, such as litter collection robots and automated monitoring systems.

In contrast with other methods in the R-CNN family, YOLO is an one-step framework based on global regression by mapping the input image directly to the prediction of bounding box locations and associated class probabilities. The main idea of YOLO is shown in Fig. 4.

In terms of the algorithm, YOLO divides the input image into an $S \times S$ grid and each grid cell is responsible for predicting the object centered in that grid cell [21]. From these grids, the network predicts the location of the bounding boxes, as well as the corresponding confidence scores that measure the likelihood of containing an object in that bounding box.

During training, YOLO uses the following sum-squared error between the predictions and the ground truth,

$$L = L_{cls} + L_{boxes} + L_{Conf} \tag{4}$$

where L_{cls} is the classification loss, L_{boxes} is the bounding box loss, and L_{Conf} is the confidence loss [25]. For more details about the YOLO architecture, please refer to [24]. For this project, we use YOLOv4-Tiny instead of the vanilla YOLOv4. It is a simplified version of YOLOv4, with less layers in the backbone and less YOLO layers. As a result, it takes significantly less time to train without compromising much of the accuracy and detection speed [26].

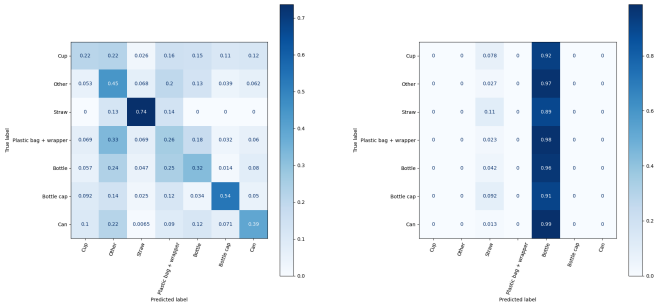
V. Experiments, Results, and Discussion

A. SVM

To assess the performance of the SVM, the main metric we used was the test accuracy, defined as the percentage of correct predictions in the test set. We also generated confusion matrices for the SVM. As can be seen in Table I and Fig. 5, we achieved the highest accuracy with the features extracted from the VGG-19 network. The confusion matrices clearly demonstrate that the HOG and Haralick feature descriptors did a poor job of classifying the images in the TACO dataset. We suspect that this is because the HOG and Haralick features across different categories were too similar.

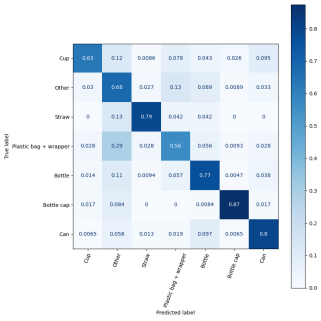
Feature Descriptor	Test Accuracy
HOG	39.52%
Haralick	17.04%
VGG-19	71.28%

TABLE I: SVM test accuracy with each feature descriptor.



(a) HOG

(b) Haralick



(c) VGG-19

Fig. 5: SVM test results with each feature descriptor

Focusing on Fig. 5c, we see that the SVM with the VGG-19 features did a reasonable job at classifying the different objects. However, it appears to get confused by the *other* category. This is not surprising because the *other* category is a conglomerate of objects in the TACO dataset that didn't fit into the categories we had chosen. The objects in the *other* category are not distinct enough for the SVM to be able to distinguish them well.

B. Mask R-CNN

For this project, we used the framework developed by Matterport [27] to implement the Mask R-CNN. To adapt this framework to our task and dataset, we made the following modifications to the training process.

1) **Transfer Learning:** Since the sample size of our training data is relatively small for such a large scale network (even after data augmentation), we chose to use transfer learning to train our Mask R-CNN model. We used the pre-trained weight of Matterport [27] as the source, which was trained with the COCO dataset [28].
2) **Data Preprocessing:** Since our dataset uses the COCO format, we needed to generate masks for the training samples before training by using the `pycocotools` API [29].
3) **Reduce Training Epoch:** We reduced the training epoch in order to avoid overfitting, adapt to the pre-trained weight we use, and shorten the training time (due to limited computational power). More specifically, we reduce the epoch trained for the backbone by 50%. The reason behind this modification is that the backbone, which is used for extracting features from images, has already been well-trained in the pre-trained weights of the source task. We also reduced the epoch for fine-tuning the network head by 20%, because our dataset has fewer categories compared to the COCO dataset.

C. YOLOv4-Tiny

We use and adapt the open-source framework `DarkNet` [24] to train the YOLOv4-Tiny model. The training process is outlined below.

1) **Transfer Learning:** We also chose to use transfer learning to train our YOLOv4-Tiny model. We used the pre-trained weights of YOLOv4-Tiny that were trained on the COCO dataset in [24].
2) **Data Preprocessing:** YOLOv4-Tiny uses a different annotation format than COCO. While COCO uses absolute coordinates to describe the location of the object within the image, YOLO uses relative coordinates. In other words, the position and dimension of an object is normalized by the scale of the image for YOLO format.
3) **Network Parameters:** Through `DarkNet`, we tuned the parameters in the top level CNN that allow YOLO to adapt to the images in our dataset. More specifically, the network size is set to 416×416 and the number of filters is set to $42 = (\#\text{ of classes} + 5) * 3$.
4) **K-Fold Cross-Validation & Early stopping:** Due to the short amount of training time required by the YOLOv4-Tiny, we were able to use 10-Fold cross-validation to determine the optimal number of iterations for training. For each fold, we obtain the weights of the model for each 1000 iterations and calculate their corresponding mAP on the validation set. Then we train the model on the entire dataset, and pick the number of iterations with the highest average mAP across the 10-folds of dataset. Based on the result, we decided to train the model with 12000 iterations, instead of the recommended 14000 iterations.

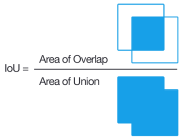
D. Results for Mask R-CNN and YOLOv4-Tiny

The performance metrics we use for Mask R-CNN and YOLOv4-Tiny are overall intersection over union (IoU) and mean average precision (mAP). The IoU for a single image is calculated as the ratio of area of overlap to area of union between the predicted bounding box and the ground truth bounding box (Fig. 6b). In this project we calculate the overall IoU by calculating the average IoU for all test images. The mAP is the average of Area Under precision-recall Curve (AUC), where precision and recall are calculated by Fig. 6a.

The mAP we are using is conventionally called mAP@50 or mAP@0.5, which stands for mean average precision with $IoU \geq 50\%$.

$$Precision = \frac{True\ positive}{True\ positive + False\ positive}$$
$$Recall = \frac{True\ positive}{True\ positive + False\ negative}$$

(a) Equation of mAP



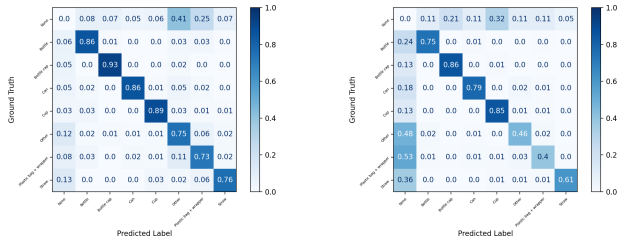
(b) Equation of IoU [24]

Fig. 6: Equations for performance metrics

In Table II, we can see that both the Mask R-CNN and YOLOv4-Tiny achieve great performance under our evaluation metrics. Both of them outperform the results of their original paper (YOLOv4-Tiny: 40.2%, Mask R-CNN: 60.0%, both trained on COCO dataset) in terms of mAP@50 [1]. The confusion matrices, shown in Fig. 7, indicate that both frameworks are able to detect the objects across all categories. Both of them, however, have relatively poor performance for the *other* category. This is not surprising because we combined several categories that have insufficient samples into *other*, and the features of this category are less distinguishable. This could be fixed, theoretically, if we had more image samples.

Models	mAP@50	IoU
Mask R-CNN	74.4%	55.3%
YOLOv4-Tiny	75.8%	57.1%

TABLE II: Performance metrics for YOLO and R-CNN



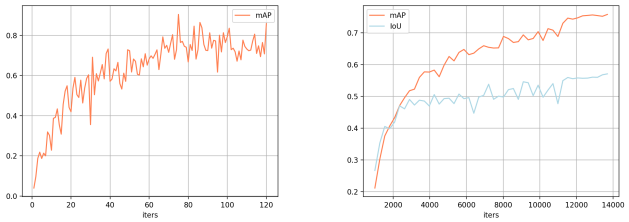
(a) Mask R-CNN (b) YOLOv4-Tiny

Fig. 7: Confusion Matrices

By using 10-fold cross-validation, we chose to train 12000 iterations—an early stop compared to the recommended 14000 iterations—to achieve the aforementioned results. From Fig. 8 we can see that the performance of both YOLOv4-Tiny and Mask R-CNN on the test set is steadily improving as the number of iterations increases, therefore no overfitting is observed[2].

[1]Although our dataset is less complicated, with only seven categories compared to 80 categories as in COCO dataset, our labeled samples are much less than the COCO dataset (1500 v.s. 120000 images).

[2]Since R-CNN is a two-stage model, we only calculate the mAP for test set for the second stage, which trains the model for object detecting.



(a) Mask R-CNN (b) YOLOv4-Tiny

Fig. 8: Performance v.s. Number of Iterations[2]

Both of the models perform state-of-the-art detection (Fig. 9), even when there are multiple objects, the objects are occluded, and when the object is camouflaged by the background. It is also worth noting that YOLOv4-Tiny is more accurate at classifying the objects, while Mask R-CNN is better at detecting the objects. This also explains why YOLOv4-Tiny has higher false negative error rate while Mask R-CNN has higher false positive error rate, which matches the results shown in the confusion matrices (Fig. 7).

(a) Mask R-CNN (b) YOLOv4-Tiny (c) Ground Truth

Fig. 9: Prediction Samples

VI. Conclusions and Future Work

YOLOv4-Tiny is a promising network for detecting and classifying recyclable objects and litter in the environment. We were able to achieve a mAP@50 of 75.8%, and an average accuracy of 82.57% across all categories.

To further improve upon these results, we would like to train a full YOLO network and the new YOLOv5. Before doing so, we would need to collect more images for the categories where YOLO under performed, like *plastic bag + wrapper*, *straw*, and *other*. Ideally we would be able to collect enough images to split the *other* category into more specific sub-categories (e.g., *paper*). Additionally, we would like to test our model on random images not sourced from the TACO dataset.

VII. Code

Project Github repository can be found at

- https://github.com/brynnemh/cs229
- Note, for the YOLOv4-Tiny and Mask R-CNN framework, we used and modified the open-source packages `DarkNet` [24] and `Mask_RCNN` [27]. All the original code is located in `/Darknet/Trash_data/` and `/Mask_RCNN/Trash/`.

VIII. Contributions

- **Gabriela Bravo-Illanes**: Dataset augmentation and image processing
- **Brynne Hurst**: Google Cloud setup and SVM implementation and training
- **Tanran Zheng**: YOLO, Mask R-CNN implementation and training

References

[1] P. Ping, E. Kumala, J. Gao, and G. Xu, "Smart street litter detection and classification based on faster r-cnn and edge computing," *International Journal of Software Engineering and Knowledge Engineering*, vol. 30, no. 04, pp. 537–553, 2020.

[2] G. Mittal, K. B. Yagnik, M. Garg, and N. C. Krishnan, "Spotgarbage: smartphone app to detect garbage using deep learning," in *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, 2016, pp. 940–945.

[3] K. Bae, K. Yun, H. Kim, Y. Lee, and J. Park, "Anti-litter surveillance based on person understanding via multi-task learning."

[4] M. Fulton, J. Hong, M. J. Islam, and J. Sattar, "Robotic detection of marine litter using deep visual detection models," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 5752–5758.

[5] J. Bai, S. Lian, Z. Liu, K. Wang, and D. Liu, "Deep learning based robot for automatically picking up garbage on the grass," *IEEE Transactions on Consumer Electronics*, vol. 64, no. 3, pp. 382–389, 2018.

[6] P. F. Proença and P. Simões, "Taco: Trash annotations in context for litter detection," *arXiv preprint arXiv:2003.06975*, 2020.

[7] T. Wang, Y. Cai, L. Liang, and D. Ye, "A multi-level approach to waste object segmentation," *Sensors*, vol. 20, no. 14, p. 3816, 2020.

[8] M. Yang and G. Thung, "Classification of trash for recyclability status," *CS229 Project Report*, vol. 2016, 2016.

[9] G. Thung, "garythung/trashnet," 2016. [Online]. Available: https://github.com/garythung/trashnet

[10] O. Awe, R. Mengistu, and V. Sreedhar, "Smart trash net: Waste localization and classification," *arXiv preprint*, 2017.

[11] A. Jung, "imgaug," 2020. [Online]. Available: https://github.com/aleju/imgaug

[12] S. Mallick, "Histogram of oriented gradients," Dec 2016. [Online]. Available: https://www.learnopencv.com/histogram-of-oriented-gradients/

[13] M.-K. Hu, "Visual pattern recognition by moment invariants," *IRE transactions on information theory*, vol. 8, no. 2, pp. 179–187, 1962.

[14] S. M. K. Bapat, S. Mallick, and K. Bapat, "Shape matching using hu moments (c++/python)," Dec 2018. [Online]. Available: https://www.learnopencv.com/shape-matching-using-hu-moments-c-python/

[15] R. M. Haralick, K. Shanmugam, and I. H. Dinstein, "Textural features for image classification," *IEEE Transactions on systems, man, and cybernetics*, no. 6, pp. 610–621, 1973.

[16] R. Chakravarti and X. Meng, "A study of color histogram based image retrieval," in *2009 Sixth International Conference on Information Technology: New Generations*. IEEE, 2009, pp. 1323–1328.

[17] D. DataTurks: Data Annotations Made Super Easy, "Understanding svms: For image classification," Aug 2018.

[18] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[19] K. Team, "Keras documentation: Vgg16 and vgg19." [Online]. Available: https://keras.io/api/applications/vgg/

[20] "About imagenet." [Online]. Available: http://image-net.org/about-overview

[21] Z.-Q. Zhao, P. Zheng, S. tao Xu, and X. Wu, "Object detection with deep learning: A review," 2019.

[22] "sklearn.linear_model.sgdclassifier." [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html

[23] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask r-cnn," 2018.

[24] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "Yolov4: Optimal speed and accuracy of object detection," *arXiv preprint arXiv:2004.10934*, 2020.

[25] J. Hui, "Real-time object detection with yolo, yolov2 and now yolov3," 2018. [Online]. Available: https://jonathan-hui.medium.com/real-time-object-detection-with-yolo-yolov2-28b1b93e2088

[26] J. Solawetz, "Train yolov4-tiny on custom data - lightning fast object detection," 2020. [Online]. Available: https://blog.roboflow.com/train-yolov4-tiny-on-custom-data-lighting-fast-detection/

[27] W. Abdulla, "Mask r-cnn for object detection and instance segmentation on keras and tensorflow," https://github.com/matterport/Mask_RCNN, 2017.

[28] 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," 2015.

[29] "pycococreator," 2020. [Online]. Available: https://github.com/waspinator/pycococreator/