Pavement Markings Detection in NYC

Final report – Data Science Capstone Project

Authors:

Moya Zhu (mz2886), Jingfei Fang (jf3420), Megan Zhou (xz2975), Ran Pan (rp3016), Zihao Zhang (zz2763)

Project Description

Pavement markings are the markings on the road used for denoting traffic characteristics. It is the most fundamental way to communicate with road users. In most cases, they are the most effective way to regulate, warn and guide traffic. Such efficient road communication can significantly improve a safe and ordered traffic environment. In addition, the color and appearance of these markings are standardized and easily identified by human beings.

However, the environment has brought erosion and abrasion to pavement markings, which will fade once they are printed. Identification of the fade has been an annoying problem since it costs time and money. Even for one specific location, the fading speed varies, which might require repetitive checking, thus bringing higher costs.

Now we have advanced aerial technology and can get high resolute images above the city. And we also have modern computer vision technology to teach AI how to identify specific objects, including all these pavement markings and the pavement color. There are already existing well-trained models like YOLO that can achieve instance segmentation, like identifying all the crosswalk markings given the satellite image as the input. However, aside from applying pre-trained models, our specific situations still required more work.

There are still lots of challenges in this area. For example, the most challenging part is, when taking the satellite image, there are different weather conditions, shadows, and leaves of trees, which significantly weaken the deep learning algorithm's performance, and this has long been a pain point in computer vision science research. In addition, training a deep learning model requires a dataset of many labeled images, which we don't have.

Project Goals

We want to detect the fading of the pavement markings for three main objects, crosswalk, green pavement, and red pavement. The green pavement is for bicycles, and the red pavement is for buses. They are the most critical markings since these pavements often fade quickly. If the comparison of fading shows a quick change, we can deduce that there might be high traffic in these lanes, and we also need to re-pave it soon.

The goal we want to achieve with this project is split into two phages:

1. Build a model that can output the location and categories of pavement marking (including crosswalk, red area for bus, green area for bicycle) when inputting an aerial image;
2. Based on the model completed in step 1, further develop our model so that when input two aerial photos show the same area, we can identify how a specific pavement marking has faded.
3. If possible, develop a program that can automatically output the degree of fade when inputting images

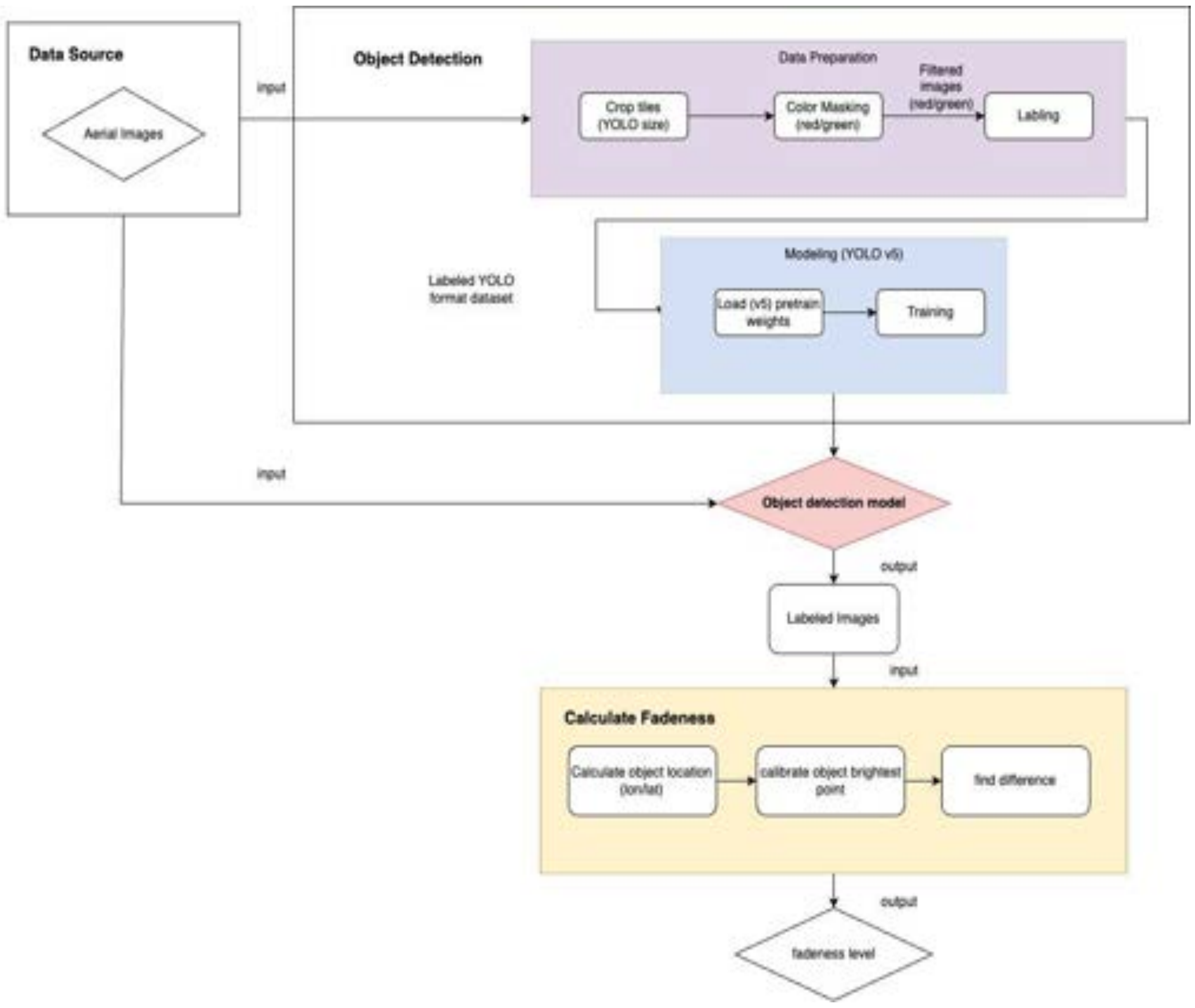
To be more specific, for task 1, we would like to build an efficient model that can accurately identify crosswalks and output the location of each crosswalk, green pavement, and red pavement. For task 2, we would like to superimpose two images taken in the exact location but at different times and define a category to qualify how faded a specific marking is. The type (to be determined) should consist of 5 or 3 degrees. This is an example of defining whiteness in 3 degrees: little white, medium white, and very white. Task 2 is essential for crosswalks and less critical concerning green and red pavement areas.

Related Work

According to the official document, “YOLO is an acronym for 'You only look once,' and is an object detection algorithm that divides images into a grid system.” YOLO is one of the most famous object detection algorithms due to its speed and accuracy. And YOLOv5 is the 5th version of this algorithm and is also the most recent algorithm of the YOLO family that achieves state-of-the-art performance on object detection.

There are countless practical applications for YOLOv5, and we simply show an example of its bounding box. (Aire, 2022)

Pipeline Flowchart



Methodology

A. Preprocessing and filtering

To build a dataset labeled red (bus) and green (bike) lanes, it’s necessary to have the data prepared in the proper size for modeling. To make the labeling process easier, we filter the samples first with an algorithm that detects a large area of color. This helps us only focus on those images with high potential to have the object we are looking for, which is much more efficient. The filtered image is further labeled with bounding boxes that bind the things we target with the associated model label format.

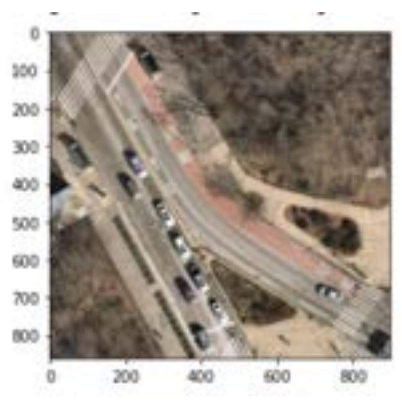
a. Cropping

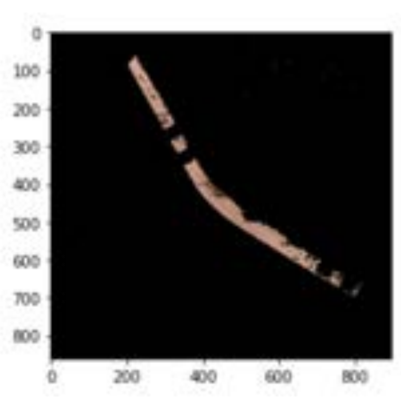
From the input of one large aerial image, we cropped the image into the size of 640*640 samples to match the YOLO v5 input. From the 38 aerial images received from our customer, we got 912 tiles with the size of 640*640 as our sample images. The samples also include edges, which fill out empty pixels to have the same size as other images. This does not influence the dataset and detection performance.

b. Color Masking

In real-world images, there are always variations in the image color values due to changes in lighting conditions, noise, shadows, etc. To overcome this problem, each image is changed into HSV (hue, saturation, brightness) color space for further analysis. Filtering out the hue channel in a particular range can give us variations from light to dark of the target color.

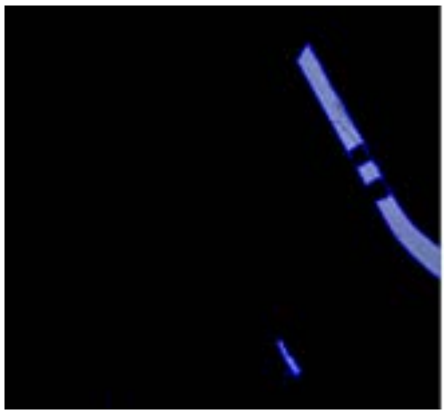
To find the hue range of the red and green colors in bus and bike lanes, we took a couple of pixels from the red and green lanes and transformed them into HSV color space to observe the value changing. After some experiments, we found that: the bike lane green is between 0.19 to 0.4 for the hue value and larger than 0.1 for the saturation channel; the bus lane red is between 0 to 0.05 for the hue value and larger than 0.3 for the saturation channel. By applying the mask range above, we can change the images into binary images by setting all other areas with the non-desired color black.





c. Area Contouring

Only knowing where are the target color is not enough to identify whether the lane exists, as there are many confusing factors, such as red trucks and green trees. We added contour to the masking shape to make this filter more accurate. In the example below, the colored area is contoured with a blue line. We then calculate the size of each contour. If the biggest contour in the image has an area larger than 500, then there is a large area of target color in the image. So we filter it out and put it into a red/green folder. Some experiments determine the threshold of 500 - if there's a target lane, the max contour area would be larger than 1000; if the lane does not exist, the area is usually below 100. So 500 is a proper value to differentiate if the potential lane exists.



This approach does not handle scenarios where large target color areas exist such as a playground with a green lane and red center. But we believe the performance is enough as a step to make the labeling easier. From 912 tiles samples, we filtered 88 images with potential green lanes and 236 images with potential red lanes. Around 60% of the filtered image do have target lanes.

B. Labeling

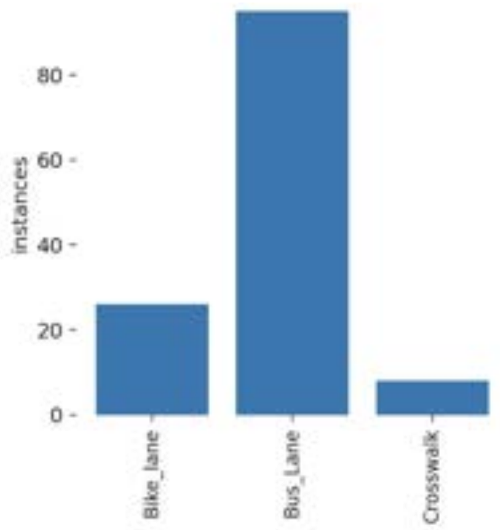
We utilized Label Studio, which is an open-source data labeling tool. With this, we could import those filters in the last step (324 images) into the interface and draw a bounding box on them. The labels we chose include bike lane, bus lane, and crosswalk. This is an example of the bus_lane label we made:

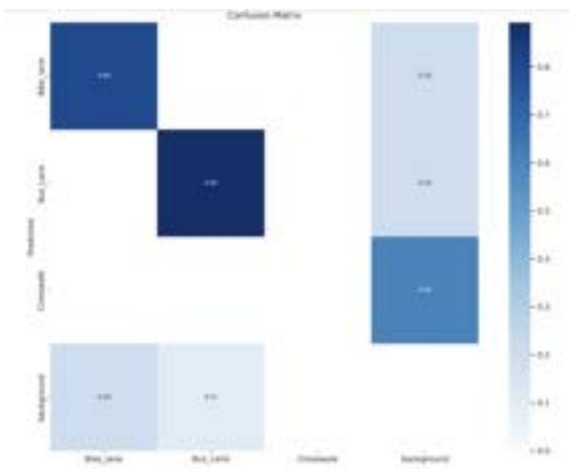
After we finished labeling. The quantity of the green lane picture we identified from those potential green lane images is 42 out of 89. The quantity of the red lane picture we identified from those potential red lane images is 112 out of 235..

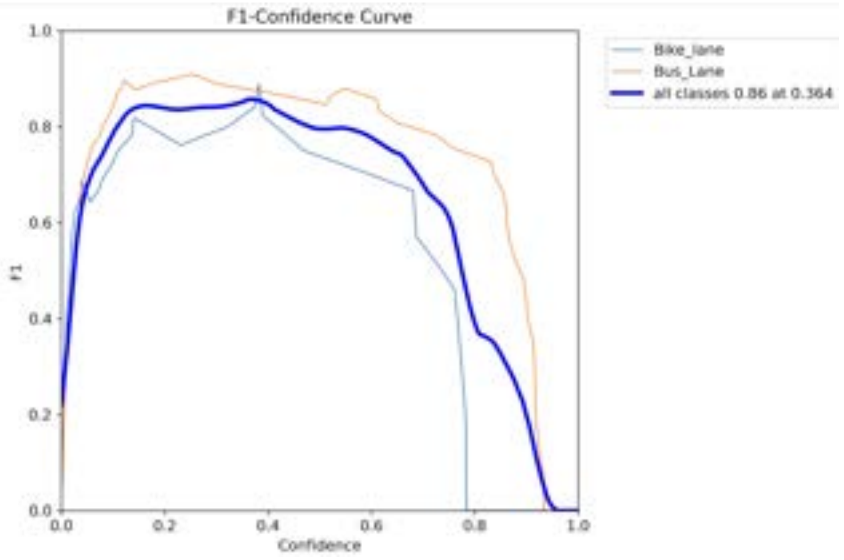
C. Object Detection

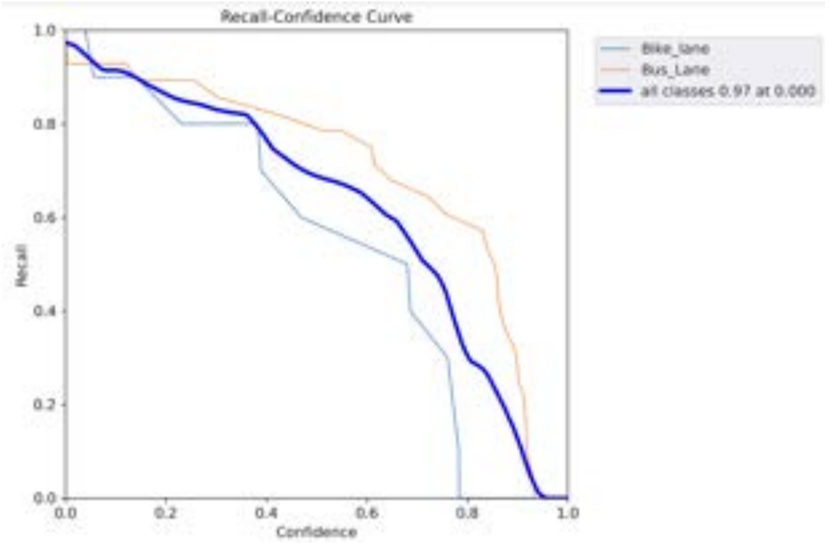
The 175 labeled images were split into 0.8 and 0.2 proportions for train and validation. YOLO v5 loaded with pre-trained weights, is used for this training. We have 3 classes as our target: Bike lane, Bus lane, and Crosswalks. We are not focusing on detecting crosswalks here as existing models already achieve high accuracy for crosswalk recognition. But we did label crosswalks if there are any in our filtered images.

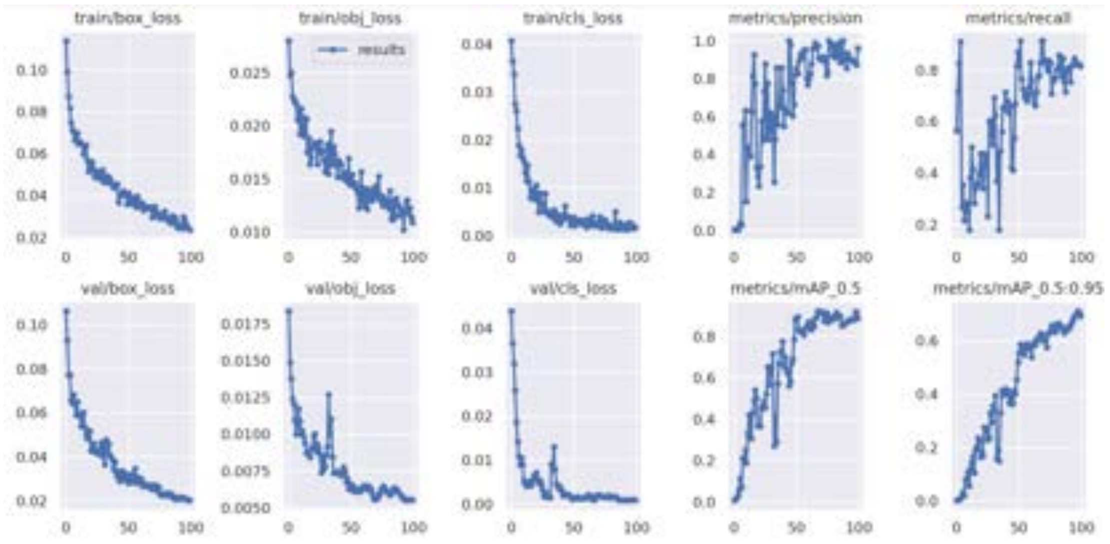
We trained the model 100 epochs with batch size = 10 on the training set. We got precision = 0.85 for bike lane, 0.92 for bus lane; recall = 0.8 for bike lane and 0.84 for bus lane. Despite the imbalance label and very small number of samples, we are already able to get a decent performance model. The training loss was still decreasing after 100 epochs. We believe with more samples and longer training time, the result can be further improved. But to capture the object and find the fade condition, we decide excessive tuning is unnecessary. Training records and result shows below:


instances
80
60
40
20
0
Bike_lane
Bus_Lane
Crosswalk




F1-Confidence Curve
Bike_lane
Bus_Lane
all classes 0.86 at 0.364
F1
Confidence
0.0
0.2
0.4
0.6
0.8
1.0


Recall-Confidence Curve
Bike_lane
Bus_Lane
all classes 0.97 at 0.000
Recall
Confidence
0.0
0.2
0.4
0.6
0.8
1.0



Model Prediction samples:

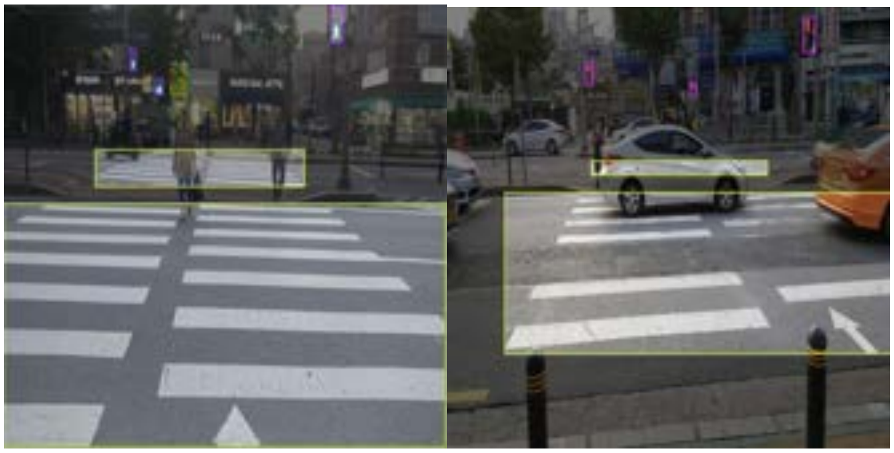
Crosswalk YOLO Object Detection

Data Source

- Korean Dataset https://open.selectstar.ai/data-set/wesee
- Size: 100+GB, but only used a small portion (63 images in total)

Data Augmentation

- Data augmentation (specifically bounding-box-level rotation -24 degrees to +24 degrees) is performed on both training and validation data. The reason is that the original dataset only contains crosswalks of a specific angle, for example:

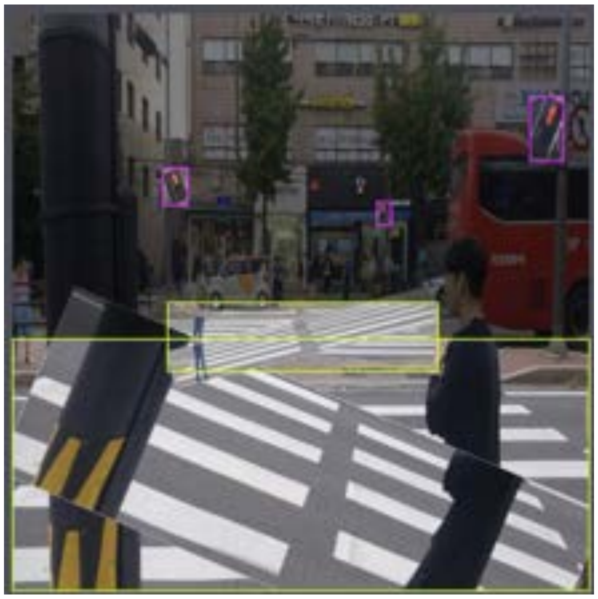


From the above examples, we can see that the crosswalks are vertical because the pictures were taken from the perspective of pedestrians. Since our aerial images contain crosswalks of all different angles, we would need augmentation on both training and validation set to mimic the behavior of aerial images.

<u>Training data after augmentation:</u> 116 images

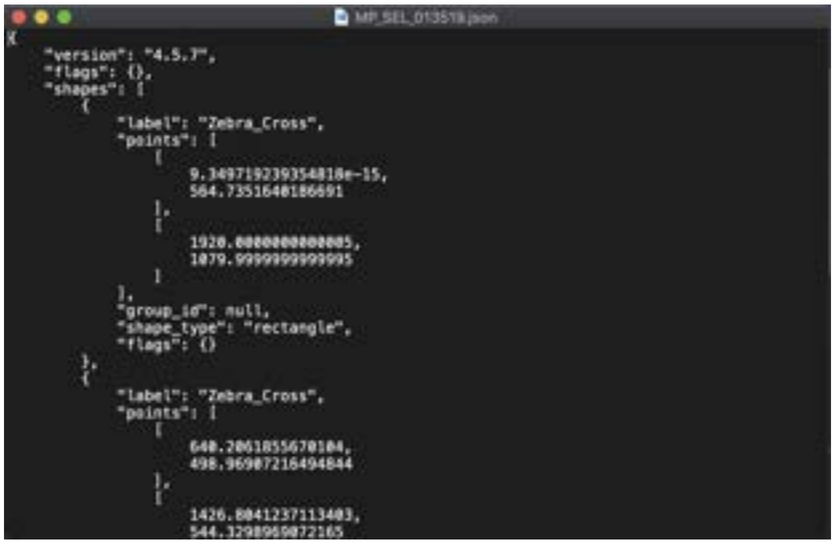
<u>Validation data after augmentation:</u> 24 images

E.g.



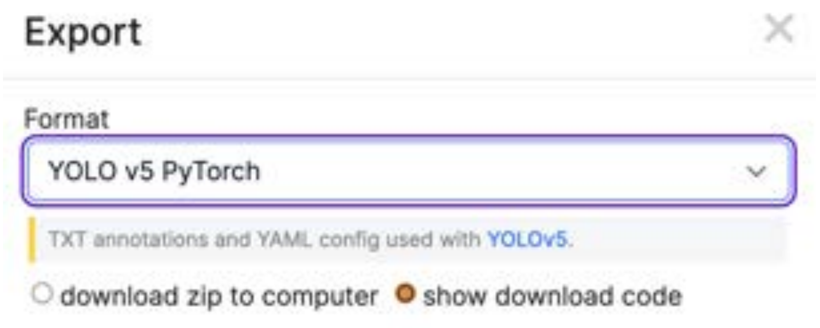
Data Formatting

- The original dataset holds bounding box coordinates of json format:



```
MP_SEL_013519.json
{
    "version": "4.5.7",
    "flags": {},
    "shapes": [
        {
            "label": "Zebra_Cross",
            "points": [
                [
                    9.349719239354818e-15,
                    564.7351640186691
                ],
                [
                    1920.0000000000005,
                    1079.9999999999995
                ]
            ],
            "group_id": null,
            "shape_type": "rectangle",
            "flags": {}
        },
        {
            "label": "Zebra_Cross",
            "points": [
                [
                    640.2061855670104,
                    498.9690721649844
                ],
                [
                    1426.8041237113403,
                    544.3298969072165
```

But the YOLO network needs the bounding box to be in YOLOv5 format, so we converted it into YOLOv5 format with Roboflow:



Export

Format

YOLO v5 PyTorch

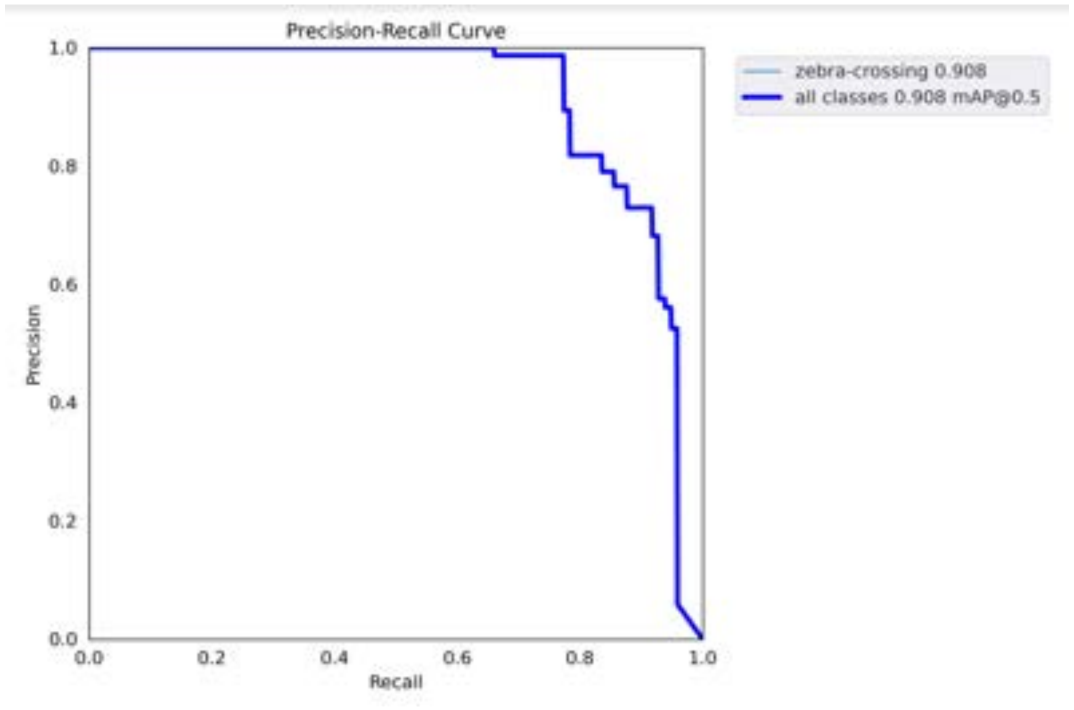
TXT annotations and YAML config used with YOLOv5.

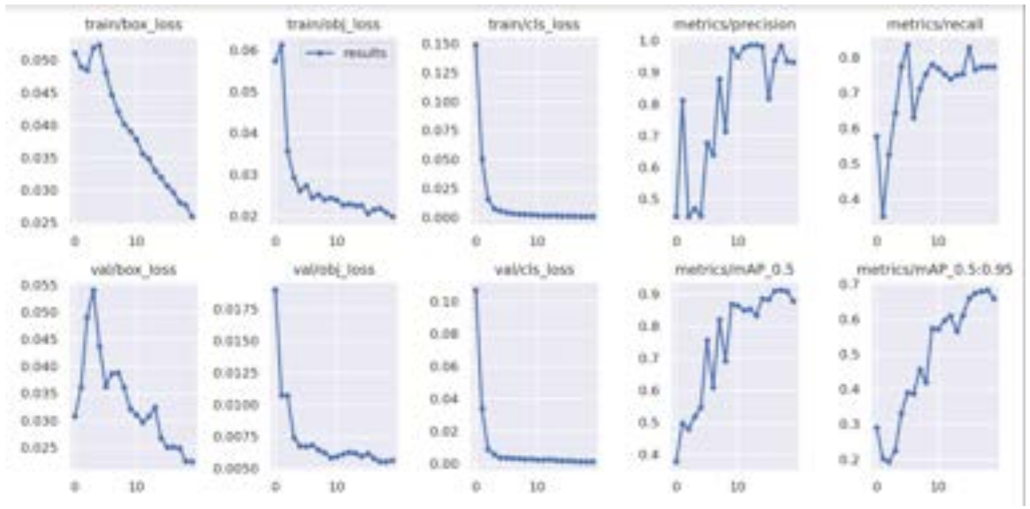
[download zip to computer] [✓ show download code]

Also, since the original dataset contains bounding boxes around green/red lights as well, we need to remove them before feeding into the YOLO network. Below is the Linux command used to remove the redundant bounding boxes:

```
find . | while read file; do sed -i '' '/^0/d'
$file; done
```

Training



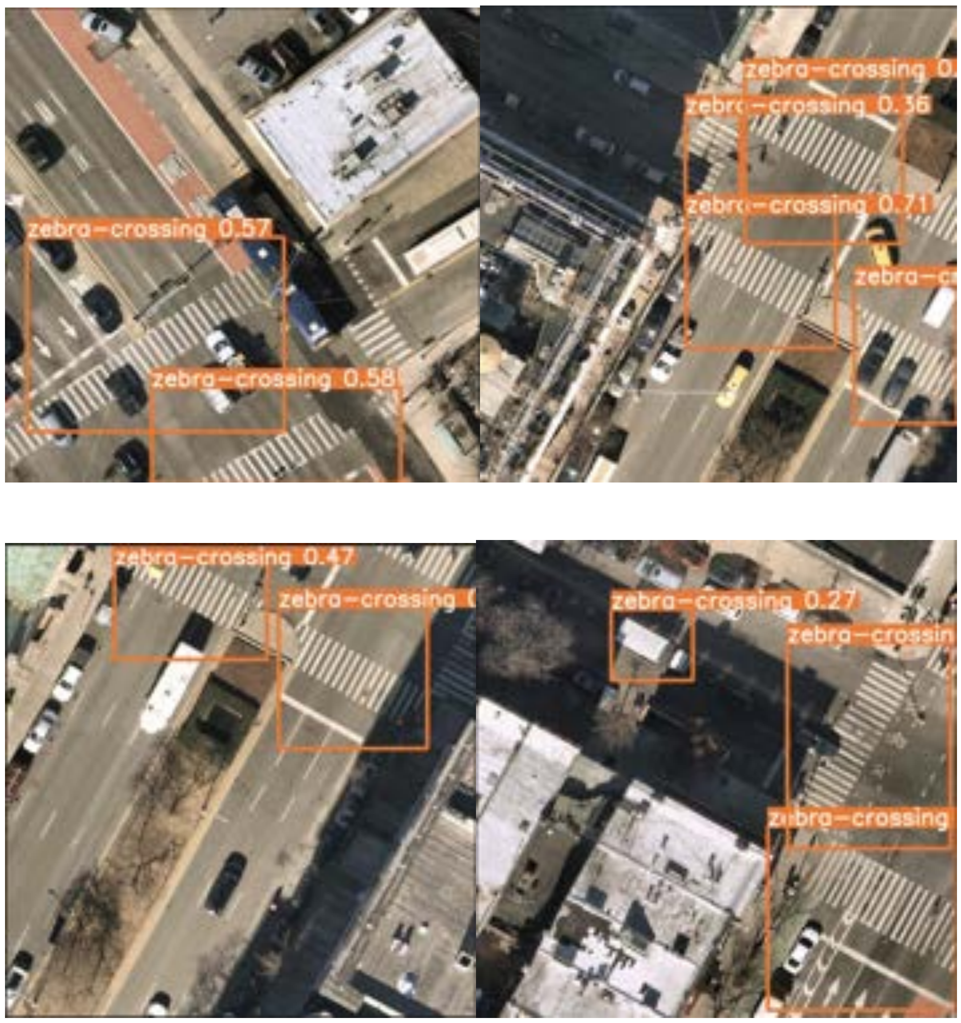


- Above is the training result. It was trained with only 20 epochs.
- The PR curve looks pretty good, with mean average precision (mAP) of 0.908.
- The validation loss also goes to nearly 0, which is very promising.

Here are some examples of the training result from the first validation batch:

<u>True label:</u> <u>Prediction:</u>

Prediction on the aerial images (examples)



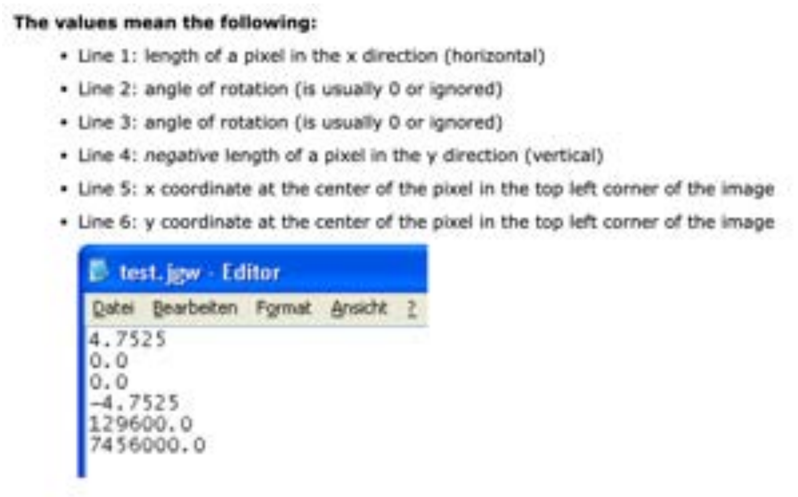
(Dilemma: we can see that the vehicle with crosswalk-like shades is also identified as crosswalk)

Coordinate Transformation

With our model already possessing the capability to output YOLO predictions from the 640*640 cropped images, we begin our next step to calculate the absolute coordinates of each pavement marking identified from the output of YOLO, on NYC plane coordinate system.

When given a new aerial image, YOLO can only generate types of pavement markings and their relative coordinates inside each cropped image. However, to know where exactly our identified pavement marking is on NYC plane coordinate system maps, we have to add a transformation module to solve this problem.

To achieve this goal, we make sure to read in the accompanying jgw file when reading one aerial image so that we know what is the scale of the image by pixel as well as the (X, Y) coordinates of the upper left pixel. Here is an example of jgw file and the meanings of each line of it:



The values mean the following:

- Line 1: length of a pixel in the x direction (horizontal)
- Line 2: angle of rotation (is usually 0 or ignored)
- Line 3: angle of rotation (is usually 0 or ignored)
- Line 4: *negative* length of a pixel in the y direction (vertical)
- Line 5: x coordinate at the center of the pixel in the top left corner of the image
- Line 6: y coordinate at the center of the pixel in the top left corner of the image

```
4.7525
0.0
0.0
-4.7525
129600.0
7456000.0
```

Having this information, when dividing the aerial image, we got to have a dictionary storing the upper left coordinates of each cropped image, and we make sure to name the tile according to its corresponding id in the dictionary. This way, after each tile, is fed into the model, and we get the output, we can simply add the multiple of the output relative coordinates and scale of the image to the pixel to the upper left coordinates to get our pavement marking the location as an absolute coordinate.

Next, we also turn out codes into a python class to automate this pipeline.

In the Coordinate.py file, we can run like this in the terminal:

```
Python coordinate.py image_jpg_path image_jgw_path label_file_folder.
```

Where the three variables mean

parameter	meaning
image_jpg_path	The directory (a file) of one aerial image
image_jgw_path	The directory (a file)of a jgw file accompanying each aerial image
label_file_folder	The directory (a file folder) of the output by YOLO, which is a file folder consisting of multiple .txt files In each .txt file, every lane records the object detection type, and its corresponding

And the output of this command is a file named crosswalk_coordinate.txt, which is the absolute coordinates for all detected crosswalk inside the given aerial image on NYC plane coordinate system.

Therefore, based on the information in the crosswalk_coordinate.txt, people can know every crosswalk on a real map.

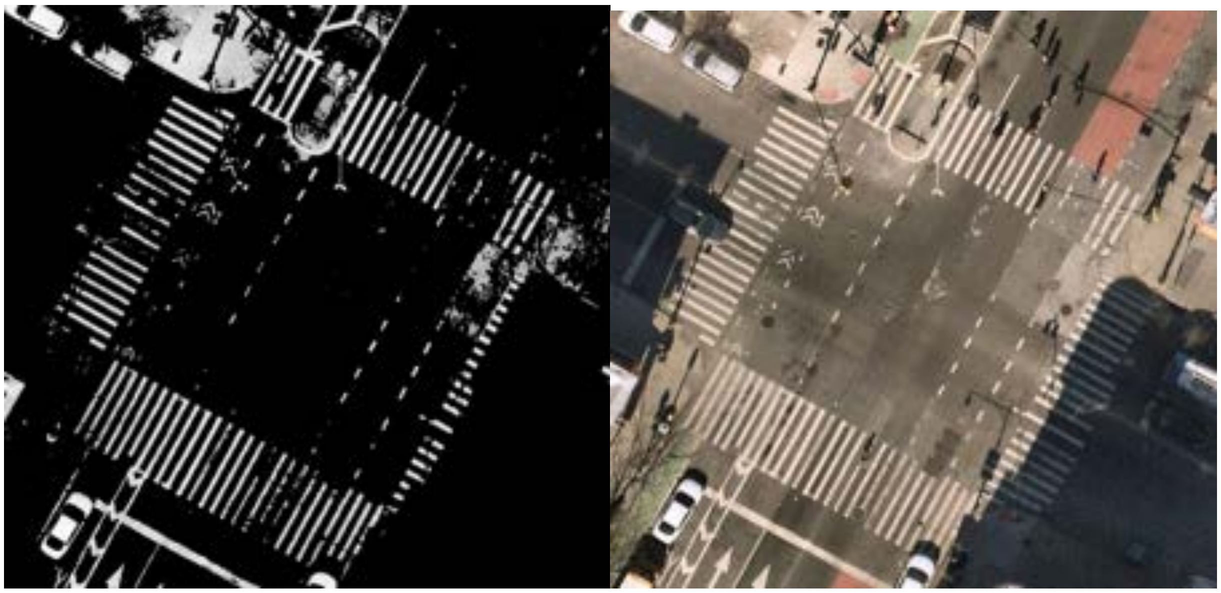
Segmentation

After having the bounding box coordinate on the map for each target pavement marking, the following step is to extract the paint-only part to compare how much the marking faded from the perfect paint. The following paragraphs explain the steps and approaches we tested in order to extract the marking.

I. Color Segmentation

Similar to Methodology b section, here we first tested using color segmentation only to extract the paint. In this step, we transformed the image to a grey level and applied a pixel color threshold. Here we selected 175 for the threshold with a max value equal to 255, which is the perfect white point. And we apply a masking on the image to only select the pixels within the selected range.

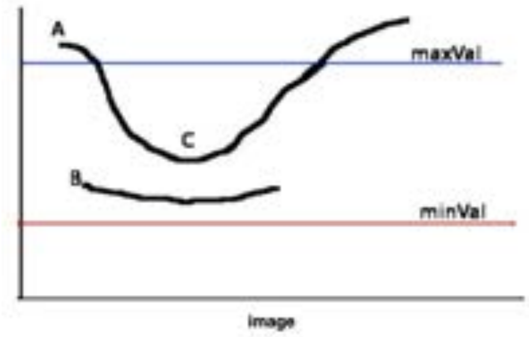
Below is an example after applying the masking (left) versus the original image (right). Ideally, we only need to focus on the crosswalks. But in the real scenario, the bounding box would contain lots of noise because the YOLO label does not support the rotation angle on the bounding box. Those noises include cars in white color, other guide markings, sidewalks, and so on. We can observe from the left image that those mentioned noises are selected. Also, This approach does not do well when there's a shadow covering the area. Because with shadow, the grey pixel level is heavily influenced, therefore, could not be extracted with the same threshold. Therefore, further steps are needed to achieve more accurate segmentation.

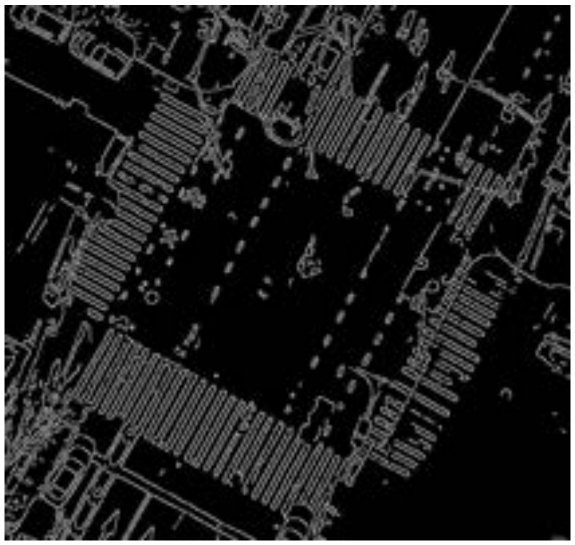


II. Contour + Shape filter

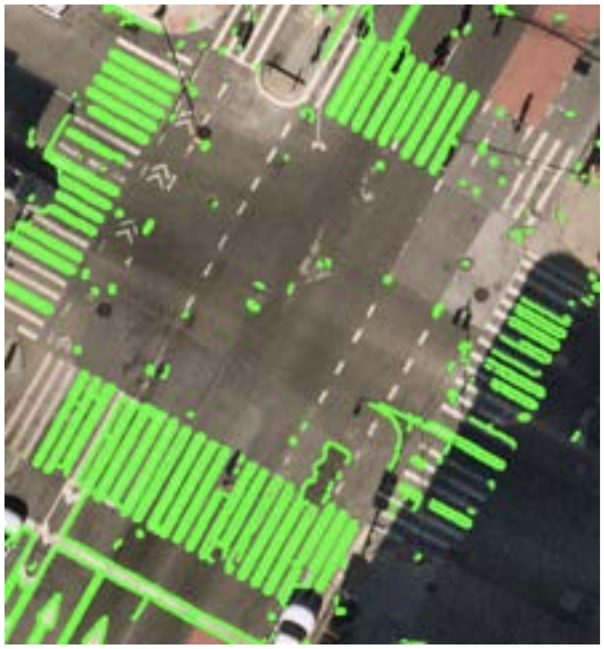
In this section, we tested an approach regarding shape in order to filter out the paint with the standard rectangle shape of the crosswalk. Here we first transform the image to grayscale and apply the GaussianBlur filter. This step is to denoise on the edges of all the objects on the image so that the contour in later steps can capture the shape more accurately.

The following step is to use the Canny Edge Detection algorithm to depict the edge of the target image. This step comes with thresholding that decides which edges we want to capture and which are not. There are two threshold values, minVal and maxVal. Any edges with an intensity gradient more than maxVal are considered edges and those below minVal are not. The figure below explains the thresholding. Edge A is above maxVal, so it can be classified as 'sure-edge'. Although C is below maxVal, as it is connected to A, it can also be considered as 'sure-edge'. We, therefore, get a full curve. Here we decide the edge threshold by trying out values in different ranges and selecting the value that best captures the crosswalk markings. The image on the right shows the result of Canny edge detection.



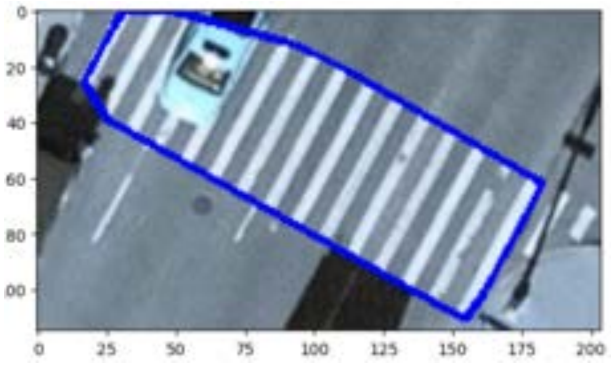


After edge detection, we can find contours to bound the shapes on the image. In this step, we selected contours in a poly shape that have an approximate arc length of around 4. This is to filter any irregular shape in the image except for simple crosswalk bars. As we can observe from the filtered contours result below, it is able to avoid cars that got selected in the previous approach and is able to pick up lots of bars under the shadow, but it is still facing problems. For example, when the bars are irregularly shaped, it cannot be picked up. This happens when another horizontal marking connects the bars, the markings come across sidewalks, or the bars are heavily flaking and cut off. We can also notice that the markings are cut off by the shadow as well. So with some tradeoffs, the approach misses lots of broken white marking but is able to avoid lots of noise. We decided further improvement on segmentation is needed.



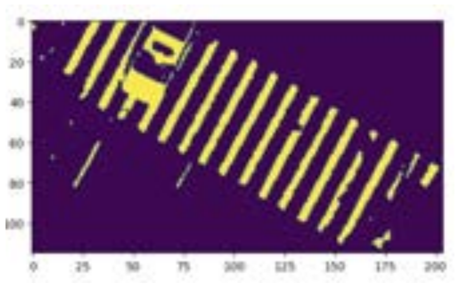
III. Convex Hull (the working solution)

Since the bounding box contains some redundant white pavement markings, we need to further limit it to contain only the crosswalk. Therefore, we tried to use convex hull (i.e. smallest polygon containing the crosswalk) to bound the crosswalk. The result looks like this: (the blue curve is the convex hull)



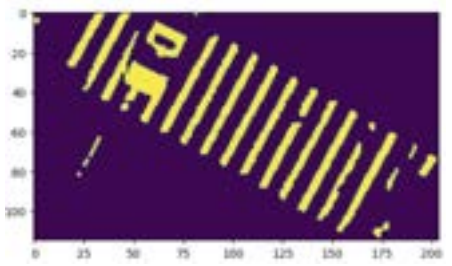
Detailed steps:

1. Filter out only the white pixels using threshold 200 to 255 with gray image.

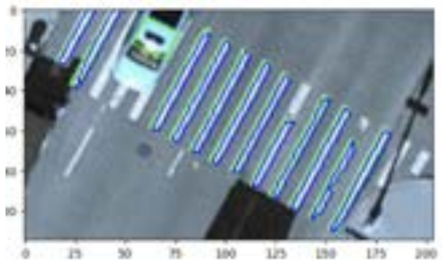


2. Denoise using **morph_open** algorithm in opencv.

   https://docs.opencv.org/4.x/d9/d61/tutorial_py_morphological_ops.html



3. Draw (blue) contours around the bars of the crosswalk. Fit (green) minimum_area_rectangle around each contour..



4. Filter out contours whose area is between 100 and 300.

   Motivation behind this step:

   Empirically, crosswalk bars usually cover an area of 100 to 300.

5. Further filter out contours that cover more than 50% of its minimum_area_rectangle.

   Motivation behind this step:

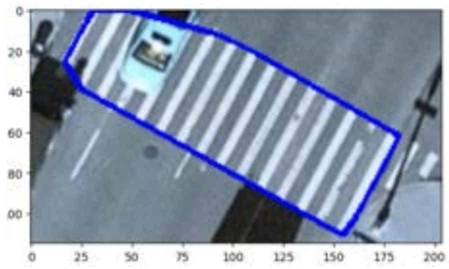
   For each crosswalk bar, the contour should be able to cover most of the area of its minimum_area_rectangle. And for some contours around some other pavement markings, it should only cover a small proportion of its minimum_area_rectangle. For example, in the following picture, the contour around the arrow marking only covers a small proportion of the green minimum_area_rectangle.

6. Further filter out contours whose minimum_area_rectangle has an aspect ratio less than 20%.

   The motivation behind this step:

   For crosswalk bars, the aspect ratio of its height to width is supposed to be very small. 20% is an empirical number after numerous experiments.

7. Fit a convex hull around the filtered-out contours.

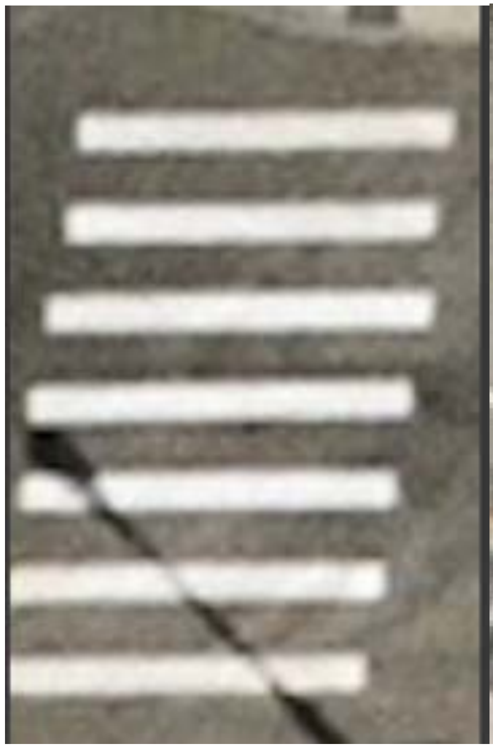


Fadedness Calculation

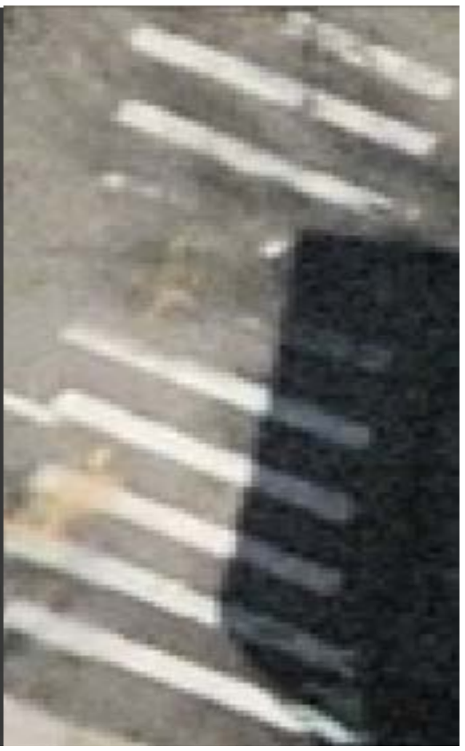
In the final section of our project, after we have computed and extracted the convex hull of the crosswalks, we can calculate the fadedness of the crosswalks based on the pixel values of the crosswalks.

Using the convex hull of the crosswalks, we can narrow the scope of crosswalks from the bounding box to the paint with the rectangle shape. From there, we can compare the fadedness of the crosswalks by calibrating the maximum pixel values of the target area to the pure white pixel values (255), and then calculate the percentage of the crosswalk's pixel values that are below a certain threshold to provide an estimation of the fadedness of crosswalks.

After numerous experimentations, we have set the threshold value to 175 as the 175-pixel value serves as a good watershed for indicating whether crosswalks faded. In the below two examples, we can observe that the left crosswalk is a perfectly conditioned crosswalk, and the right one has faded.



98.7%



87.5%

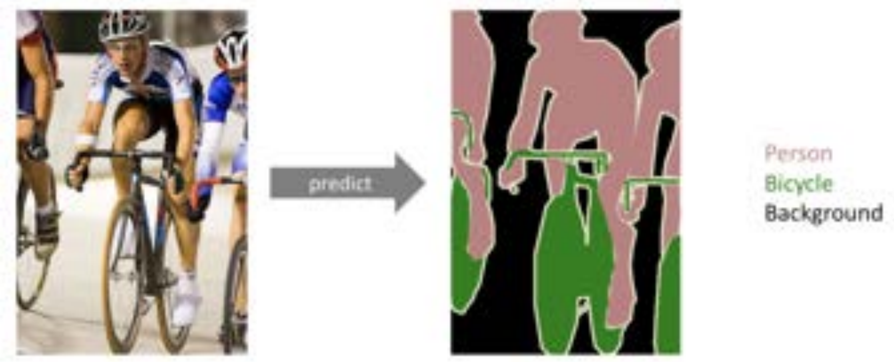
Challenge & Next step

In the end, our project achieved the two-staged goal we set in the beginning: recognizing pavement markings and their fadeness when looking at aerial images. In retrospect, we want to discuss our work and its breakthroughs and limitations.

With our first part of the project (object detection of pavement markings,) we stopped when realizing with 200 labeled images of the bike lane and bus lane, we've already got a decent performance of 0.82 -0.9 precision and recall. Therefore we did not spend more time on improving the model. However, this model can be further improved by longer training time, parameter tuning, and training with more data (we have only around 30 aerial images).

We noticed that lots of the flaking part that is completely missing on the pavement markings could not be captured with the current segmentation approach. This is mostly because we do not have a reference or original shape of the markings. An approach would be to compare two images with different time spans, superimpose them, in order to further compare. This requires having multiple images, preferable in the same angle but different time, in order to compare.

Another approach would be using semantic segmentation models. This is an approach to classify pixels into one or more classes which are semantically interpretable. Categorizing the pixel values into distinct groups can be done with CNN models. In this way, the paint layer would be extracted from the beginning of the pipeline. Below is an example of semantic segmentation on objects. However this model requires a more complicated labeling process (group pixels) and model training approach. We have only seen success in large objects so far and performance on delicate markings application should be further investigated.



References

Aire, L. G. (2022, March 14). *The practical guide for Object Detection with YOLOv5 algorithm*. Towards Data Science. Retrieved October 20, 2022, from https://towardsdatascience.com/the-practical-guide-for-object-detection-with-yolov5-algorithm-74c04aac4843