Github

https://github.com/fyng/gmm_seg_classifier.git

Introduction

In this project, we aim to identify objects in the scene through training a machine learning mode. The task is formulated as *semantic segmentation* – partitioning the scene into objects by assigning a label to every pixel in an image.

Problem Statement

Given an array of pixels (a 2D image), segment the pixels containing orange cones.

Approach

Here, I solve the simpler problem of classifying a given pixel into ['orange cone', 'not orange cone'], independent of spatial location and the surrounding pixel context. This is as opposed to e.g. a convolutional neural network approach, where the convolution kernel explicitly mixes information from a small neighbourhood of pixels

My pipeline consists of the following: 1. A pixel classifier: given a pixel, represented as an array of [R-value, G-value, B-value], return a probability of the pixel being an orange cone versus not. The pixel classifier returns a probability map over the image, which I then threshold to produce a binary classification mask. 2. A cleanup pipeline: use a series of dilation and erosion operations to clean up artifacts from the binary classification mask from step 1. The triangle shape is then fitted to the remaining contours. 3. Introducing geometric constrains: further refine the contours from step 2 by removing contours that does not abide by the known height-to-base ratio of the cone. 4. Distance estimation: using the pinhole camera model approximation, estimate the distance from the camera to the cone using the known dimensions of the cone.

1.Pixel classifier

I implemented two models for pixel classification: a simple Gaussian model and a Gaussian Mixture Model (GMM). Both models utilize multidimensional Gaussians with 3 variables, for the RGB channels respectively. *I choose the RGB space since it uses all available information in the image..*

The orange cones are manually segmented from the images and the models are trained on the collection of all orange cone pixels. Separately a background model is trained on the background pixels. During training, the model learns to estimate P(x|y, theta), where x is the pixel in RGB, y is the label, and thera is the model's internal representation.

During the fitting, the model parameters are used to calculate the posterior probability of P(y|x). The posterior probability is derived using the cone model for p(cone | x), the background model for p(background | x), and the prior p(cone) was empirically derived from relative abundance of cone pixels in the training set. In particular, I choose not to set p(background | x) = 1 - p(cone | x) as the multivariate Gaussian has a long tail and the normalized probability density function asssigns a small value to each point.

Running the pixel classifier on a test image returns a probability mask of the same size as the image. Each pixel is assigned a value [0,1] corresponding to its p(cone).

Gaussian Model

The Gaussian model is parameterized by its mean and variance. To fit the model, the mean and variance can be directly taken over the training data points. During inference or fitting, the mean and variance

Gaussian Mixture Model

The GMM tries to fit 2 Gaussian components to the data. I choose `n_components=2` based on visual inspection of the images, which are taken indoors (lower-lighting) and outdoors (bright lighting). I believe that `n_components=2` is a good tradeoff between representation power and computation time.

To jointly fit the 2 Gaussians, I implemented the EM-algorithm. The EM algorithm consists of two main steps that are repeated iteratively until convergence: * Expectation Step (E-Step): Given the current estimates of the model parameters, calculate soft cluster assignment (*responsibility*) of each data point to the two components in the GMM. This is done by calculating the probability of the data for each cluster under the multivariate Gaussian, and normalizing the probabilities to 1. * Maximization Step (M-Step): Update the model parameters (means, covariances, mixing coefficients) using the *responsibilities* computed in the E-Step. This maximizes the likelihood of the observed data point under the current estimates of the cluster memberships.

2.Cleanup pipeline

The probability mask from step 1 is thresholded to create a binary classification mask for cones. *The threshold is determined by visually inspecting the binary classification mask from the training dataset.* A potential future extension is to optimize the threshold empirically using a Intersection-over-Union (IoU) metric between the training dataset segmentation and the classification mask.

The binary classification mask then undergoes a series of dilations and erosions to clean up small regions, which are likely misclassifications. The erosion uses a small square kernel to remove small regions, and the subsequent dilation restores the eroded border in the remaining large objects.

Next, a polygon contour is fitted around each remaining region of the binary mask (*shown in blue*). A bounding box around the polygon contour is derived. This bounding box should encompass the entire cone, and enable further geometric constrains.

3.Geometric constraints

The height and base width of the triangle can be directly taken from the bounding box. We know that the cone should be 17 inches by 7.5 inches, and I use that ratio to further reject objects which have the same color profile, and therefore were misclassified as orange cones (e.g. orange podium, sofa). Specifically, I impose that the height / width of the bounding box must be within a tolerance threshold of the orange cone. The threshold is also determined by visually inspecting the training dataset.

4.Distance estimation

The pinhole camera model describes the following equation:

$$\frac{\text{object size}}{\text{image size}} = \frac{\text{distance to object}}{\text{focal length}}$$

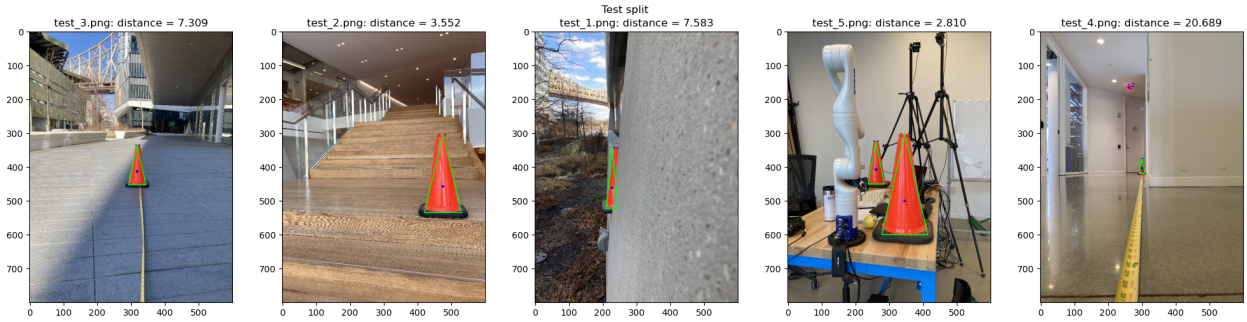
We know that the cone is 17 inches in height and 7.5 inches in width, and we also have a set of segmented training images with cone height and width in pixels, as well as the known distance to the object. Therefore, the focal length can be calculated. Here, I found a **focal length of 50 pixels**.

To calculate the distance of a cone in the test set, I calculate the distance to object as $\frac{\text{object size} \times \text{image size}}{\text{focal length}}$. The height and base width of the cone is used to estimate the distance independently, then averaged to derive the reported distance estimate.

Results

Test dataset:

```
ImageNo = [3], CentroidX = 318.000, CentroidY = 414.000, Distance = 7.309
ImageNo = [2], CentroidX = 475.000, CentroidY = 458.000, Distance = 3.552
ImageNo = [1], CentroidX = 228.000, CentroidY = 462.000, Distance = 7.583
ImageNo = [5], CentroidX = 261.000, CentroidY = 408.000, Distance = 7.336
ImageNo = [5], CentroidX = 346.000, CentroidY = 501.000, Distance = 2.810
ImageNo = [4], CentroidX = 300.000, CentroidY = 404.000, Distance = 20.689
```



Train split:

```
ImageNo = [12], CentroidX = 429.000, CentroidY = 458.000, Distance = 7.044
ImageNo = [17], CentroidX = 343.000, CentroidY = 470.000, Distance = 7.594
ImageNo = [4], CentroidX = 163.000, CentroidY = 247.000, Distance = 4.814
ImageNo = [29], CentroidX = 310.000, CentroidY = 508.000, Distance = 6.831
ImageNo = [2], CentroidX = 51.000, CentroidY = 506.000, Distance = 14.567
ImageNo = [2], CentroidX = 362.000, CentroidY = 437.000, Distance = 28.504
ImageNo = [2], CentroidX = 162.000, CentroidY = 440.000, Distance = 8.526
ImageNo = [7], CentroidX = 590.000, CentroidY = 726.000, Distance = 61.607
ImageNo = [7], CentroidX = 454.000, CentroidY = 554.000, Distance = 8.930
ImageNo = [10], CentroidX = 490.000, CentroidY = 402.000, Distance = 18.229
ImageNo = [21], CentroidX = 316.000, CentroidY = 362.000, Distance = 9.828
ImageNo = [27], CentroidX = 97.000, CentroidY = 407.000, Distance = 28.590
ImageNo = [31], CentroidX = 296.000, CentroidY = 578.000, Distance = 9.565
ImageNo = [20], CentroidX = 180.000, CentroidY = 454.000, Distance = 7.202
ImageNo = [5], CentroidX = 2.000, CentroidY = 411.000, Distance = 79.567
ImageNo = [5], CentroidX = 132.000, CentroidY = 385.000, Distance = 23.684
ImageNo = [22], CentroidX = 480.000, CentroidY = 302.000, Distance = 6.347
ImageNo = [26], CentroidX = 482.000, CentroidY = 414.000, Distance = 29.792
ImageNo = [24], CentroidX = 321.000, CentroidY = 407.000, Distance = 11.562
ImageNo = [3], CentroidX = 68.000, CentroidY = 496.000, Distance = 13.384
ImageNo = [15], CentroidX = 425.000, CentroidY = 385.000, Distance = 2.865
ImageNo = [14], CentroidX = 327.000, CentroidY = 527.000, Distance = 14.583
ImageNo = [16], CentroidX = 439.000, CentroidY = 457.000, Distance = 23.986
```

Validation split:

```
ImageNo = [1], CentroidX = 313.000, CentroidY = 421.000, Distance = 30.804
ImageNo = [6], CentroidX = 534.000, CentroidY = 448.000, Distance = 8.369
ImageNo = [18], CentroidX = 235.000, CentroidY = 501.000, Distance = 4.609
ImageNo = [13], CentroidX = 428.000, CentroidY = 437.000, Distance = 5.224
ImageNo = [19], CentroidX = 427.000, CentroidY = 389.000, Distance = 14.355
```

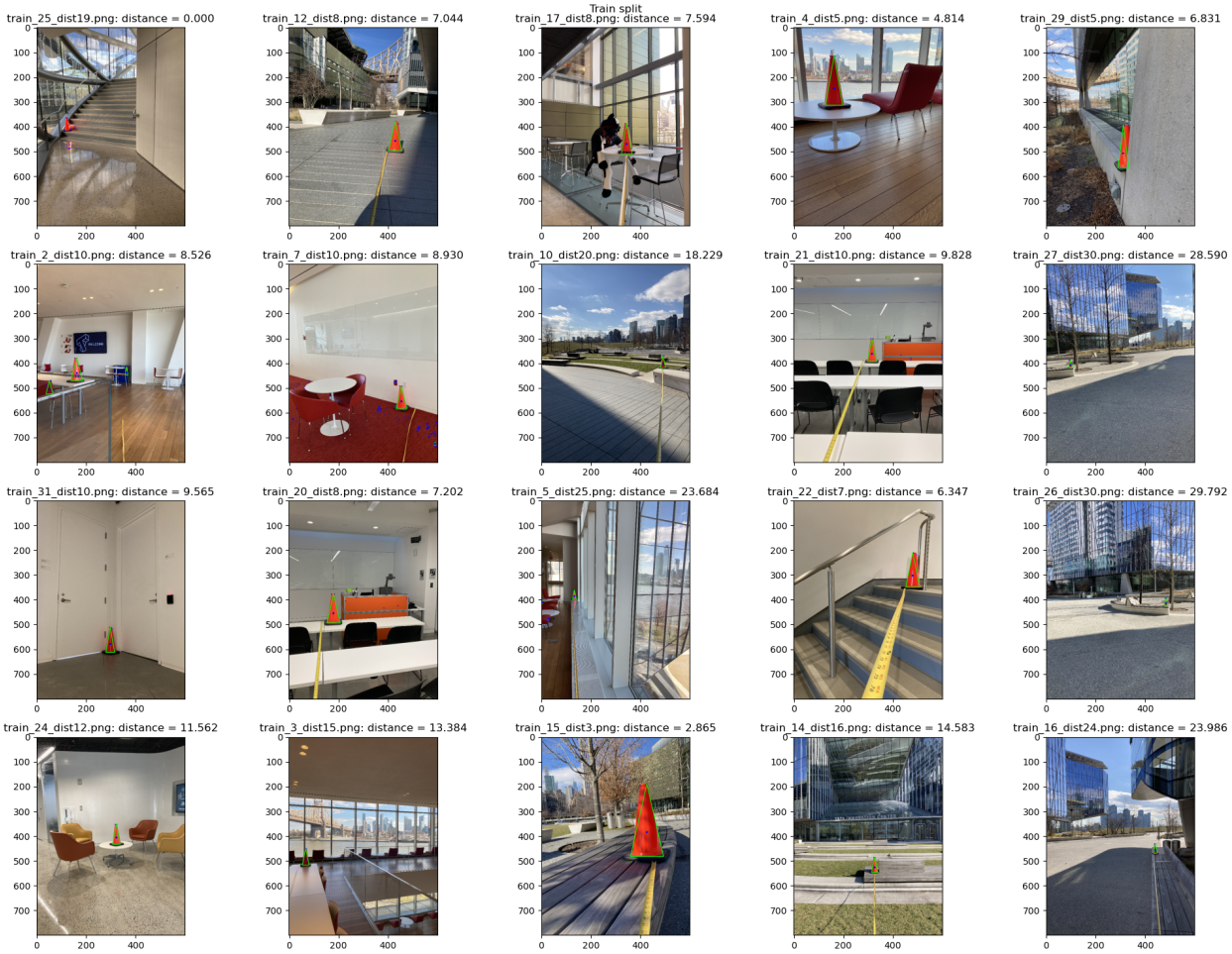


Figure 1: Train Visualization

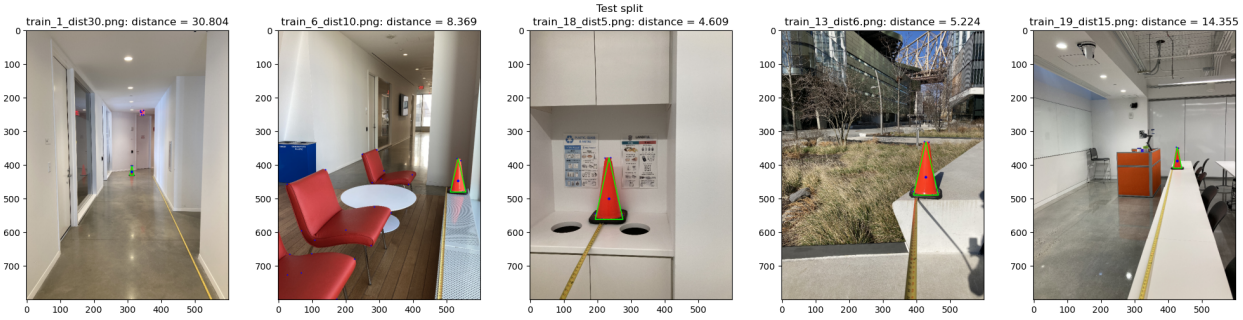


Figure 2: Validation Visualization