Problem 1

RANSAC, standing for Random Sample Consensus, is a method to estimate parameters of a mathematical model from a set of observed data that contains outliers, when outliers are to be accorded no influence on the values of the estimates.

Generally speaking, the algorithm can be described as follows:

1. Randomly select two points from samples.
2. Get the hypothesized model based on the samples randomly chosen. In this case, it is a line which passes these two points.
3. Calculate the distance between other points and the line respectively. Count the number of samples whose distances are less than the threshold of distance.
4. If the number counted in last step is equal to or greater than one specific threshold for sample counts, we assume this line as the final fitting result. Else, go back to step 1 (Of course, the points picked up in the next round should not be the same one as before)

In this case, it’s required to set 2 thresholds, respectively for the distance as well as the sample count. I set the threshold of the point count as 70% of the total as well as 0.125 for the threshold of distances. It’s also necessary to set a parameter called rounds, making the calculating process done when it tries ‘rounds’ times without meeting the sample count threshold requirement. The best of the ‘rounds’ times trying is chosen as the final result.

For more details, you may take the codes attached as references, which is with detailed comments. The result is shown as follows. The light blue line represents the fitting result. The blue dots are the inner samples as well as the red dots being the outer samples.
Problem 2

AdaBoost, short for Adaptive Boosting, is a method combining a set of weak classifiers into a weighted sum that represents the final output of the boosted classifier.

In this given case, weak classifiers are predefined as vertical or horizontal lines and to train AdaBoost is to get the weighted combination of some lines. The steps can be described briefly as follows:

1. Generate 100 horizontal lines and vertical lines in the range of x and y values of the point samples.
2. Set the initial weights of samples same as one of the total numbers, or say evenly.
3. Use them to test samples respectively. Find the best one getting the highest correct rate, or say accuracy and determine the direction, whether the left zone of the line (or zone below the line, in horizontal cases) is taken as positive or negative samples.
4. Calculate alpha for this weak classifier as \( \frac{1}{2} \ln(\text{correct rate}/(1 - \text{correct rate})) \);
5. Modify weights for point samples. Let’s say the original weight for one specific point sample is \( w \). If the weak classifier can classify the sample correctly, the weight should be modified as \( w \times \exp(-\alpha) \), as well as \( w \times \exp(\alpha) \) in the case of incorrect classification. In this way, the weight for the sample correctly classified is decreased, while the one for the sample classified incorrectly is increased. After all weights are modified, they should be normalized.
6. To calculate the prediction value using AdaBoost, it’s required to sum \( \alpha \times \text{prediction} \) by each weak classifier and take the sign of the value as prediction value. Calculate prediction value for each point. Times the prediction value, the actual value and the weight for this sample respectfully for each point and sum them as the accuracy of the boosted classifier.
7. If the accuracy is acceptable, or say greater than the preset threshold, the training process is done. Else, go back to step 3.

In this case, ten samples and their values are given. Following the steps given before, three lines, two vertical ones and one horizontal line are found out, which are \( x = 93.7575757575758 \), \( y = 85.2222222222222 \) and \( x = 258.848484848485 \). For horizontal lines, points in the zone below it is taken as positive while left zone positive, for the vertical line. The alpha values for each weak classifier are \( 0.423648930193602 \), \( 0.649641492065131 \) and \( 0.922913345249166 \). Since the threshold is set as 1, the result boosted classifier classifies all samples perfectly.

I also implement the function to predict any points using this boosted classifier, in which x and y are required. You may call it as follows:

```
predictByAdaBoost(classifierValueSet, classifierTypeSet, classifierDirectionSet, alphaSet, 136, 275);
```

The prediction result will be shown in the console window, as:

```
The prediction value for point(136,275) is -1
```

For more details, you may take the codes attached as references, which is with detailed
comments. The result is shown as follows. The red points represented by add sign is samples with positive value, while negative valued sample as blue circle. The lines represent weak classifiers.
Problem 3

\[
\text{var}(\omega^T X) = \frac{1}{n-1} \sum_{i=1}^{n} \omega^T (x_i - \mu)(x_i - \mu)^T \omega = \omega^T C \omega, \text{where } \mu = \frac{1}{n} \sum_{i=1}^{n} x_i
\]

C is the covariance matrix, which is a p-dimensional real symmetrical matrix. So we may assume the eigen-vectors \(\alpha_1, \alpha_2, \ldots, \alpha_p\) and corresponding eigen-value \(\lambda_1, \lambda_2, \ldots, \lambda_p\).

\(\alpha_1, \alpha_2, \ldots, \alpha_n\) are orthogonal basis as the eigen-vectors of covariance matrix can be taken as the orthogonal basis. So, we get \(\omega = e_1 \alpha_1 + e_2 \alpha_2 + \cdots e_p \alpha_p\). Put it back to the equality above, we may get:

\[
\text{var}(\omega^T X) = \omega^T C \omega = \sum_{i=1}^{p} (e_i \alpha_i)^T C \sum_{i=1}^{p} (e_i \alpha_i) = \lambda_j \sum_{i=1}^{p} \sum_{j=1}^{p} e_i e_j \alpha_i^T \alpha_j = \lambda_j \sum_{j=1}^{p} e_j^2
\]

Also, it may be proven that:

\[
\omega^T \omega = \left( \sum_{i=1}^{p} e_i \alpha_i \right)^T \left( \sum_{j=1}^{p} e_j \alpha_j \right) = \sum_{i=1}^{p} \sum_{j=1}^{p} e_i e_j \alpha_i^T \alpha_j = \sum_{j=1}^{p} e_j^2 = 1
\]

Here, we may assume that \(\lambda_1 > \lambda_2 > \cdots > \lambda_p\). So, \(\text{var}(\omega^T X) = \lambda_j \leq \lambda_1\), when \(e_1 = 1\), while others equal to 0. Thus, \(\omega = \alpha_1\), the value of \(\text{var}(\omega^T X)\) reach the max when the \(\lambda_j\) reach the max value.

If \(\omega \neq \alpha_1\), we denote \(\omega\) as \(\omega'\). \(\omega'\) is orthogonal to \(\alpha_1\) since \(\omega\) is orthogonal to \(\omega'\). We get:

\[
\omega^T \alpha_1 = e_1^T \alpha_1^T \alpha_1 = 0
\]

\(e_1' = 0\) since \(\alpha_1^T \alpha_1 \neq 0\). So, we get:

\[
\text{var}(\omega'^T X) = \sum_{j=2}^{p} e_j'^2 \lambda_j = \sum_{j=2}^{p} e_j'^2 \lambda_j \leq \lambda_2 \sum_{j=2}^{p} e_j'^2 = \lambda_2
\]

when \(e_2 = 1\), while the others are 0. Thus, \(\omega = \alpha_2\) is the eigen-vector of C corresponding to its second largest eigen-value.

Provident done.
Problem 4

The sigmoid function is:

$$h(x) = \frac{1}{1 + e^{-x}}$$

The derivative of it can be transformed as:

$$h(x)' = \frac{e^{-x}}{(1 + e^{-x})^2} = \frac{1}{1 + e^{-x}} \cdot \frac{1 + e^{-x} - 1}{1 + e^{-x}} = h(x) \cdot (1 - h(x))$$

To calculate the gradient of the cost function for logistic regression:

$$\nabla_\theta J(\theta) = -\sum_{i=1}^{m} y_i \cdot \frac{h_\theta(x_i)(1 - h_\theta(x_i))x_i}{h_\theta(x_i)} + (1 - y_i) \cdot \frac{-h_\theta(x_i)(1 - h_\theta(x_i))x_i}{1 - h_\theta(x_i)}$$

$$= -\sum_{i=1}^{m} y_i (1 - h_\theta(x_i))x_i + (1 - y_i)(-h_\theta(x_i)x_i)$$

$$= \sum_{i=1}^{m} (h_\theta(x_i) - y_i)x_i$$

So, the gradient of this cost function is $\nabla_\theta J(\theta) = \sum_{i=1}^{m}(h_\theta(x_i) - y_i)x_i$.

Provident done.
Problem 5

In this case, YoloV3 object detection network is applied.

The task is to recognize and locate two specific kinds of products in real time, so that accuracy as well as recognizing speed should be considered. 94 original images are taken as training images, labeled by labelImg. Also, Gaussian blur is applied to these images so as to improve the generalization ability. 20% of the images are chosen as validation set, while others for training.

In training process, pre-trained weights for COCO is used and the network is further trained for 100 epochs. After training process, the video is taken apart into images by frame and the frame set images are detected by the trained network. In the end, the output images are combined into the result video, which is attached to the submission package.

Generally speaking, the result of detection and localization is satisfying. Almost all target objects are recognized and localized successfully. The only exception is a non-target object is detected as one target between frame 653 to frame 663. (Frankly speaking, the packaging color of this non-target object is totally same to one of target objects, so that the mistake is reasonable for me.) No target object is missed.

In the implementation process, one difficulty is to collect dataset since the packaging style changes and not so many supermarkets nearby sales the target yellow instant noodles. I believe the result may get better if a better dataset is provided in training process.

Please take the attached video as reference.