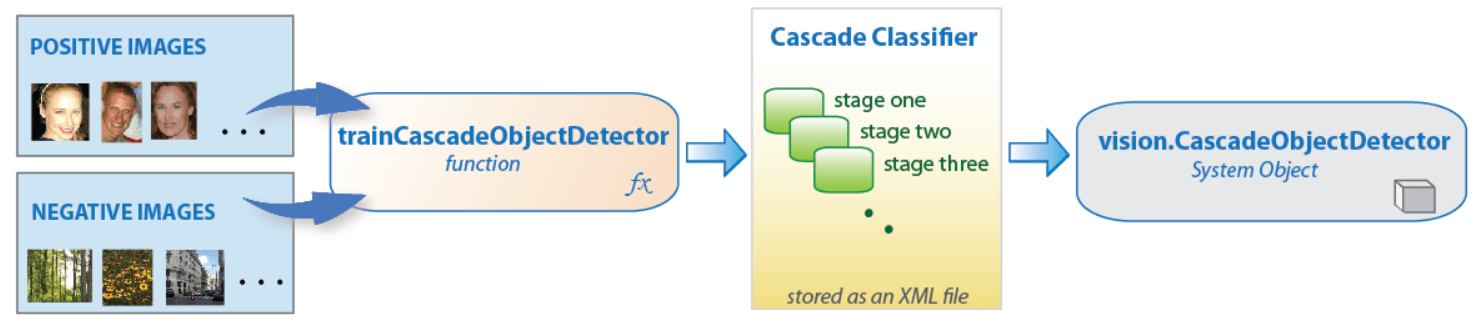
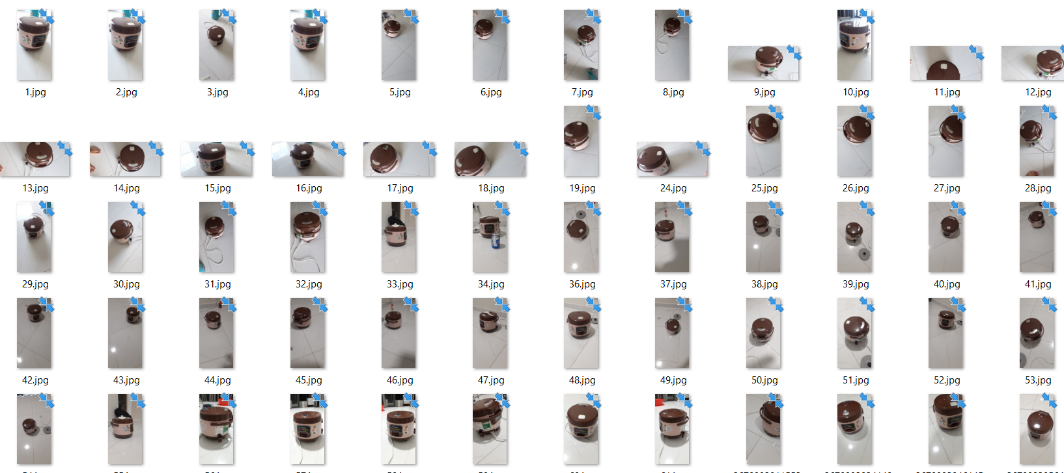
**Train Cascade Object Detector**

**Step by step to create a model training cascade object detector**



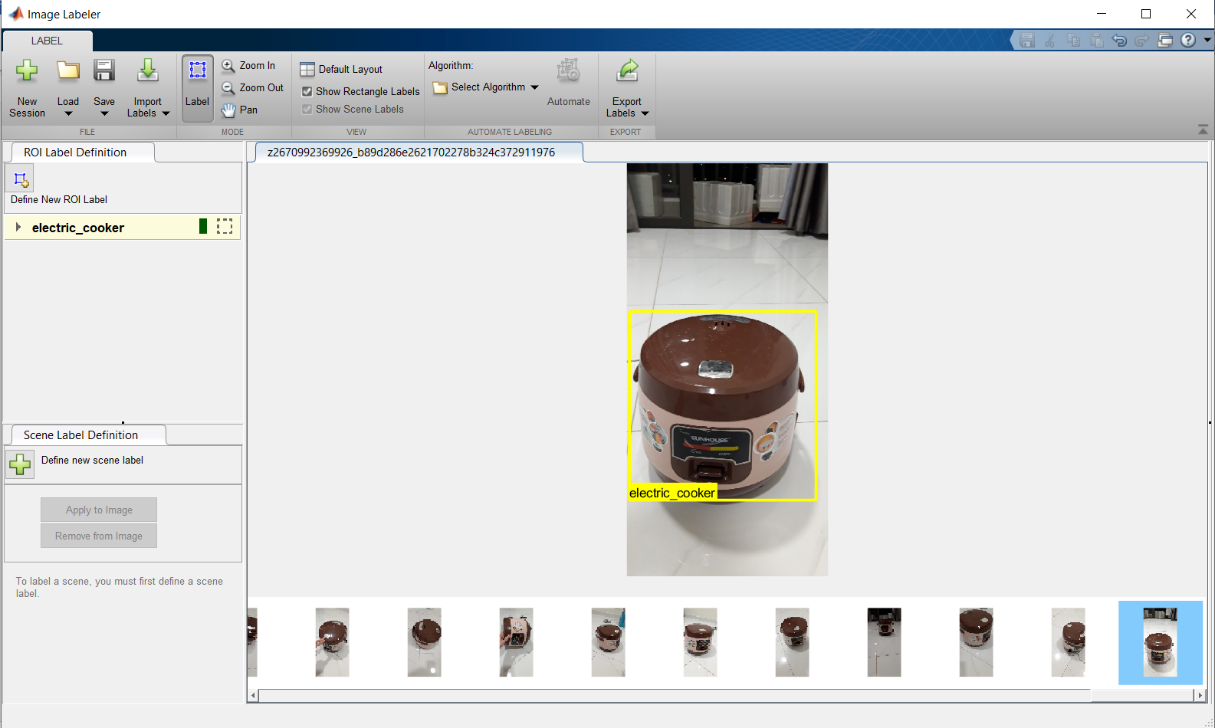
1. **Collect Dataset( positive and nagative images)**

* Collect images, save them into 2 files is positive and negative. In this case, we prepare 93 positive images and 1110 negative images.

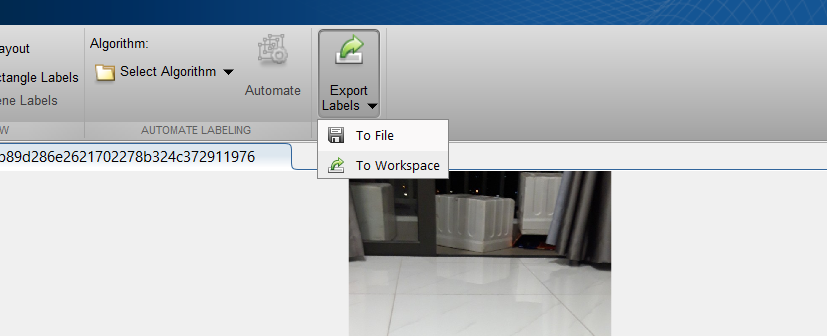


*Positive Image*

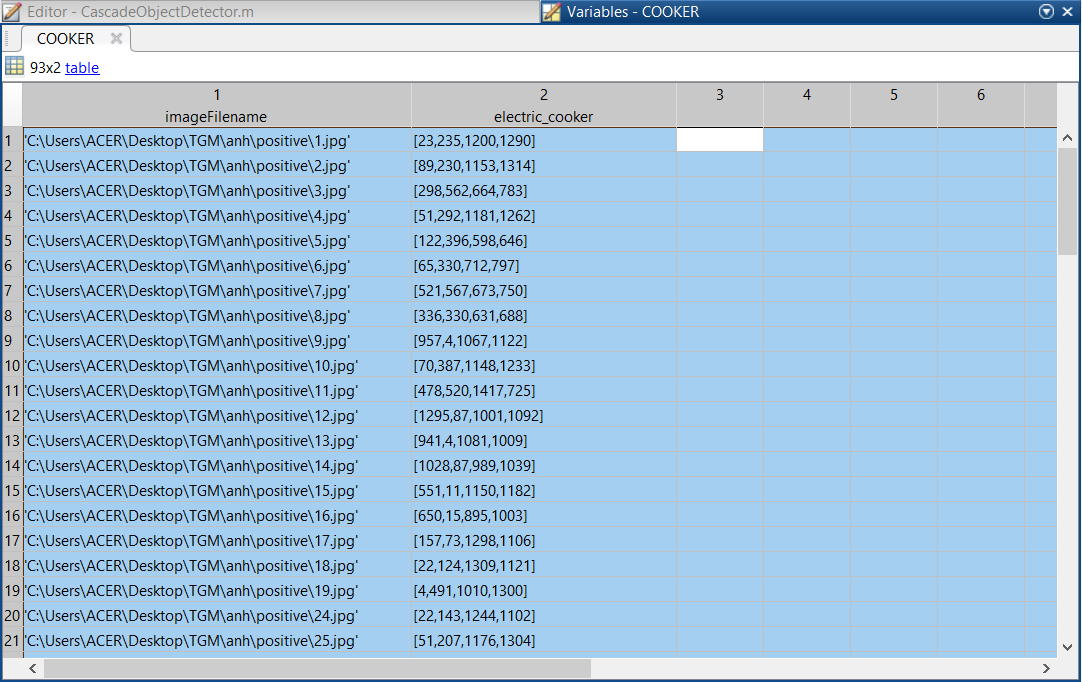
* Use the Image Labeler or Video Labeler app to label objects of interest with bounding boxes. The app returns a groundTruth object. Use the objectDetectorTrainingData function to obtain a table from the object to use for positiveInstances. The function automatically determines the number of positive samples to use at each of the cascade stages. This value is based on the number of stages and the true positive rate. The true positive rate specifies how many positive samples can be misclassified.



* We can also specify positive samples manually in one of two ways.
  + Specify rectangular regions in a larger image. The regions contain the objects of interest. The other approach is to crop out the object of interest from the image and save it as a separate image or specify the region to be the entire image.
  + Generate more positive samples from existing ones by adding rotation or noise, or by varying brightness or contrast.
* Object, in this case, is the cooker. After the labeling is done, Export Labels into Workspace with table format. In this step, It saved with the “COOKER” name.

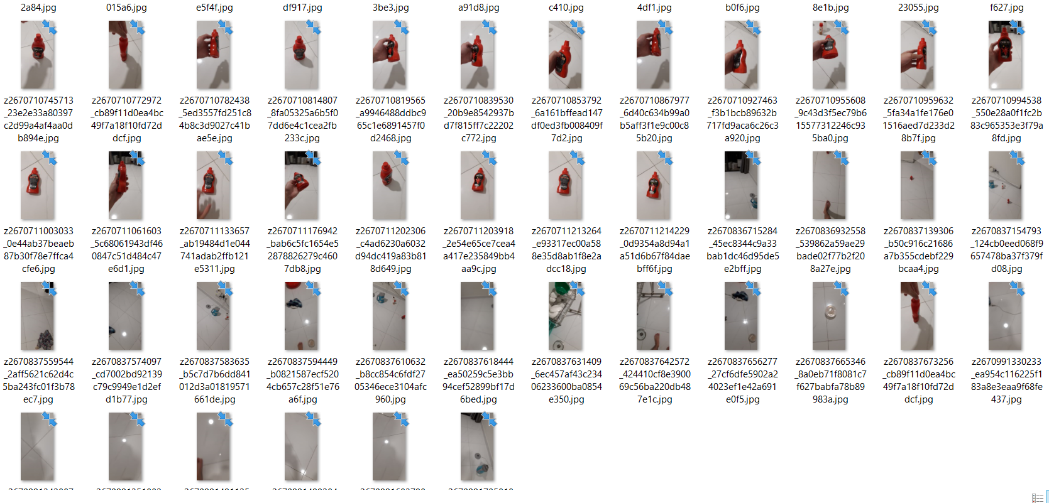


*Export labels file to workspace*



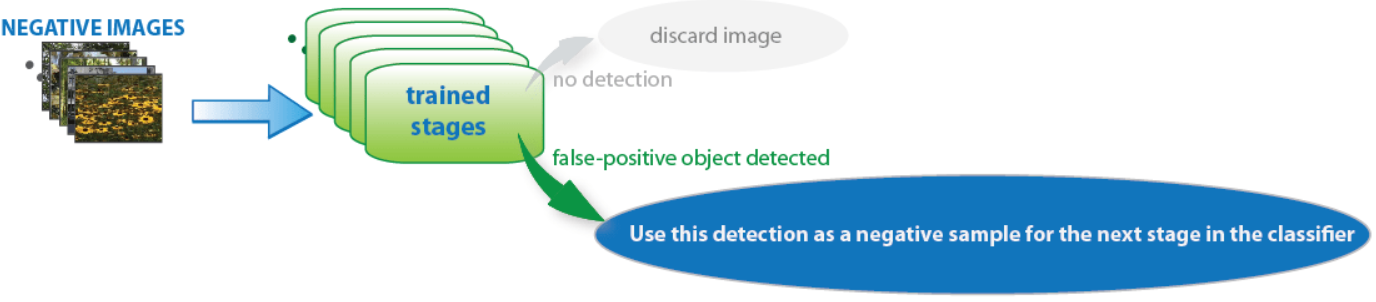
*File in workspace*

* Positive samples, specified as a two-column table or two-field structure.
  + The first table column or structure field contains image file names, specified as character vectors or string scalars.
  + The second table column or structure field contains an M-by-4 matrix of M bounding boxes. Each bounding box is in the format [x y width height] and specifies an object location in the corresponding image.



*Negative Image*

* Negative images, specified as an ImageDatastore object, a path to a folder containing images, or as a cell array of image file names. Because the images are used to generate negative samples, they must not contain any objects of interest. Instead, they should contain backgrounds associated with the object.



* As more stages are added, the detector's overall false positive rate decreases, causing generation of negative samples to be more difficult. For this reason, it is helpful to supply as many negative images as possible. To improve training accuracy, supply negative images that contain backgrounds typically associated with the objects of interest. Also, include negative images that contain nonobjects similar in appearance to the objects of interest.

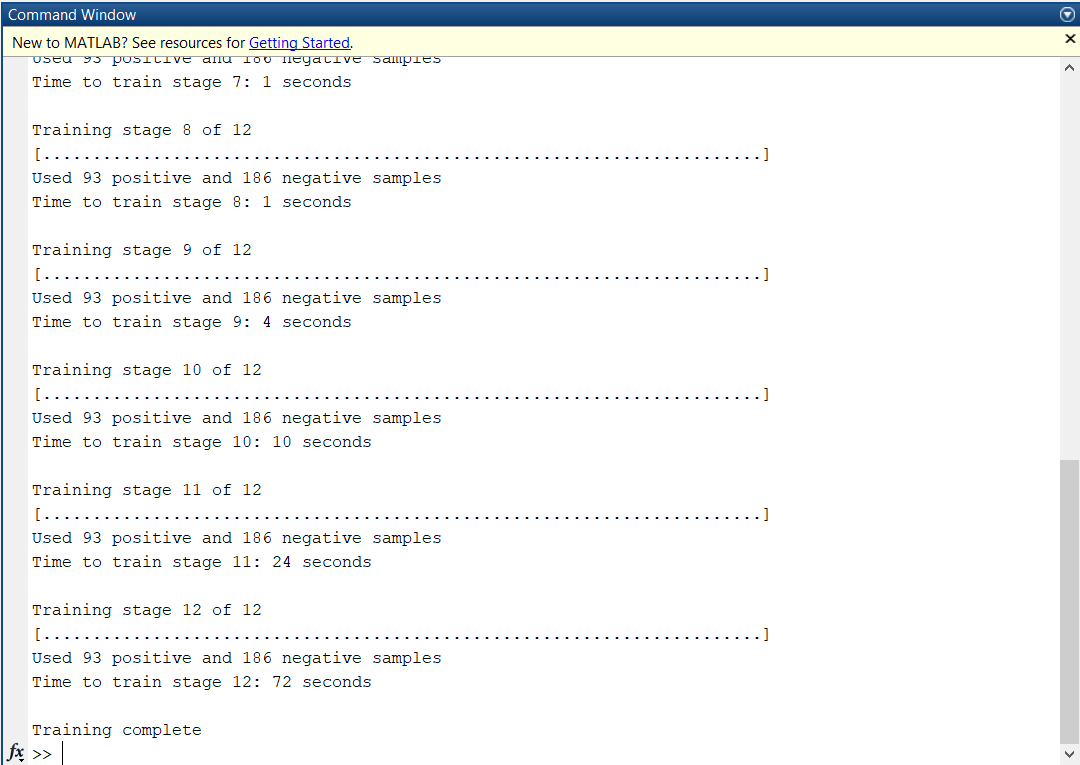
1. **Training( Use cascadeObjectDetector Function)**

* Setting parameter: Select the function parameters to optimize the number of stages, the false positive rate, the true positive rate, and the type of features to use for training.
* Considerations when Setting Parameters:

|  |  |
| --- | --- |
| **Condition** | **Consideration** |
| A large training set  (in a thousand images) | Increase the number of stages and set a higher false positive rate for each stage. |
| A small training set | Decrease the number of stages and set a lower false positive rate for each stage. |
| To reduce the probability of missing an object. | Increase the true positive rate. However, a high true positive rate can prevent you from achieving the desired false positive rate per stage, making the detector more likely to produce false detections. |
| To reduce the number of false detections. | Increase the number of stages or decrease the false alarm rate per stage. |

* There is a trade-off between fewer stages with a lower false positive rate per stage or more stages with a higher false positive rate per stage. Stages with a lower false positive rate are more complex because they contain a greater number of weak learners. Stages with a higher false positive rate contain fewer weak learners. Generally, it is better to have a greater number of simple stages because at each stage the overall false positive rate decreases exponentially
* Number of cascade stages to train, specified as the comma-separated pair consisting of 'NumCascadeStages' and a positive integer. Increasing the number of stages may result in a more accurate detector but also increases training time. More stages can require more training images, because at each stage, some number of positive and negative samples are eliminated. This value depends on the values of FalseAlarmRate and TruePositiveRate. More stages can also enable you to increase the FalseAlarmRate
* Set the false positive rate (False Alarm Rate) and the number of stages, (NumCascadeStages) to yield an acceptable overall false-positive rate. Then we tune these two parameters experimentally. In this case, after trying with a lot of stages and false alarm rates. We find out with NumCascadeStages = 12, FalseAlarmRate = 0.35 have the best result.
* Matlab code:

|  |
| --- |
| % Load image and bouding box of COOKER  positive\_ins = COOKER;  % Specify folder with positive images  pos\_dir = fullfile('C:\Users\ACER\Desktop\TGM\anh\positive');  addpath(pos\_dir);  % Specify folder with nagetive images  neg\_dir = fullfile('C:\Users\ACER\Desktop\TGM\anh\negative');  % Train the detector  trainCascadeObjectDetector('model.xml', positive\_ins, neg\_dir,...  'NumCascadeStages',12,'FalseAlarmRate',0.38); |



* Training a good detector requires thousands of training samples. Large amounts of training data can take hours or even days to process. After completely training, we have a file with extension “.XML” format. In this case we created with “model.xml” name.

1. **Test model cascade object detector.**

* Test with video:

|  |
| --- |
| vid = VideoReader('test/2.mp4');  detector = vision.CascadeObjectDetector('model.xml');  while hasFrame(vid)  vf = readFrame(vid);  bbox = step(detector,vf);  detectedImg = insertObjectAnnotation(vf,'rectangle',bbox,'cooker');  imshow(detectedImg);  end |

* Result: Display the detected cooker.

