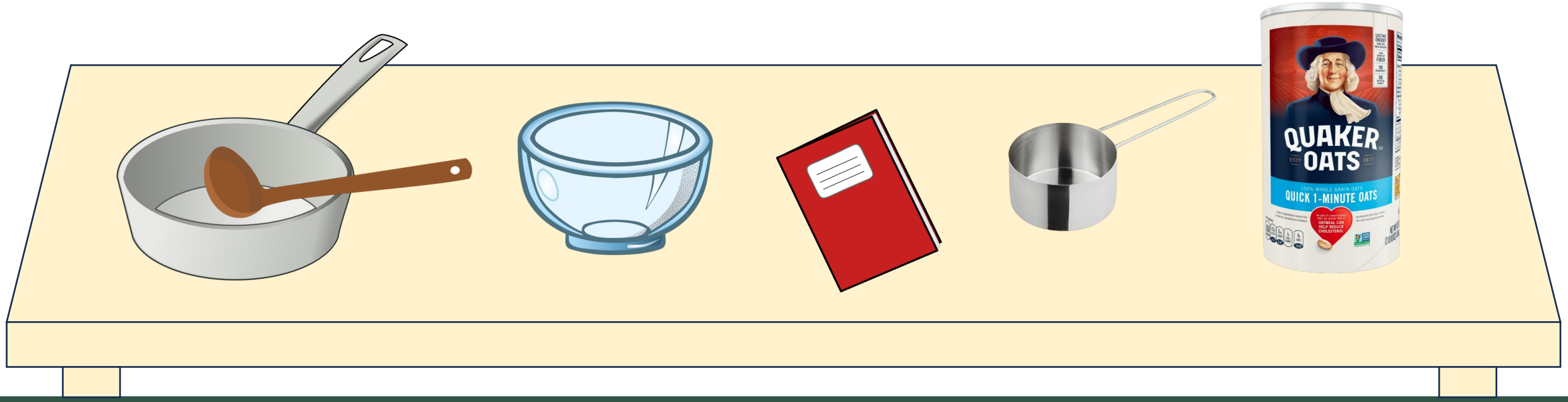
Computer Vision Tutorial

Ruiqi Wang

9/7/2023

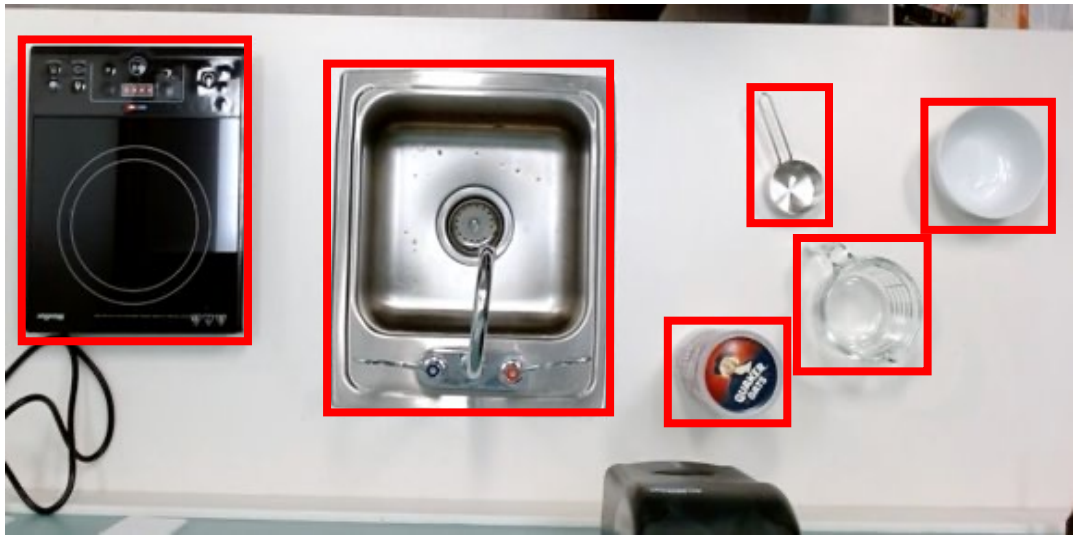
Recap: Project Goals

- To build an IoT system that can detect **if the required items are correctly taken on the table**.
  - ***True*** if all the required items are on the table without distractors.
  - ***False*** if any required item is missing or any distractor is on the table.
    - **What is missing and what is not needed?**



Recap: Localization using CV

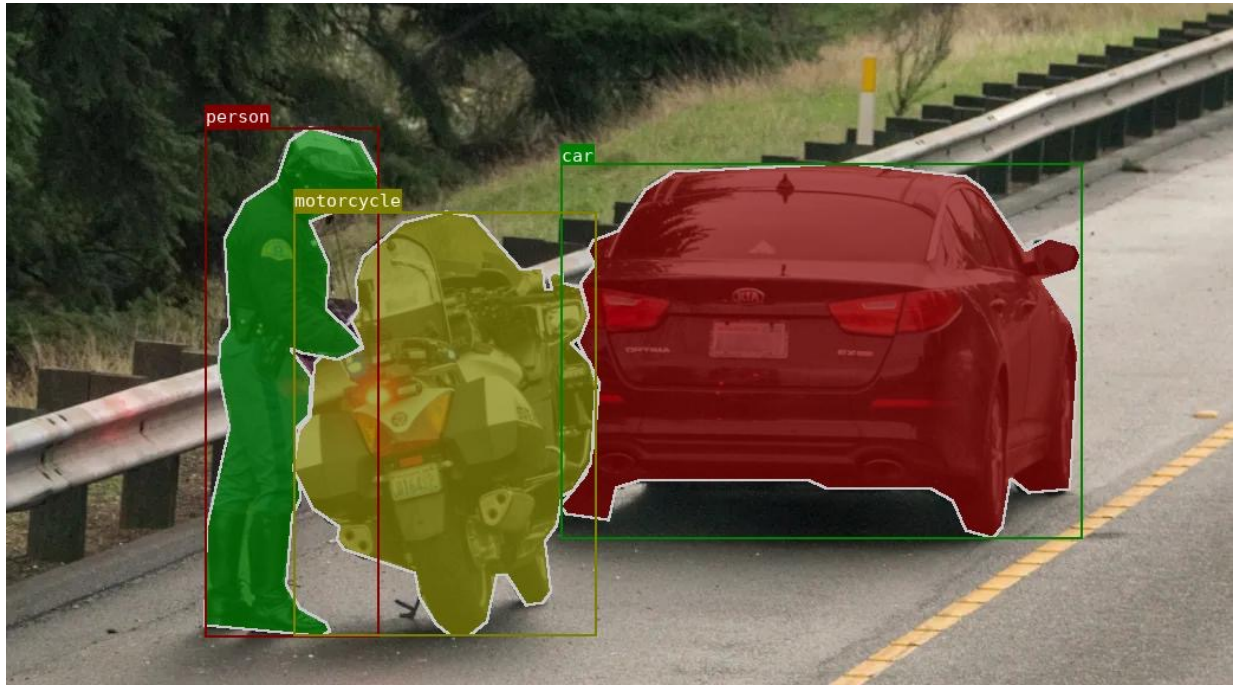
- With a camera, we can localize the items using CV and deep neural networks (DNNs).
  - **Related CV topics:** Object detection, object tracking



Computer vision

- Computer vision is a field of computer science that focuses on enabling computers to identify and understand objects and people in images and videos.

- We will not cover the topics in “CSE 559A-Computer Vision” but some hands-on experience related to IoT and course project.



Topics Today

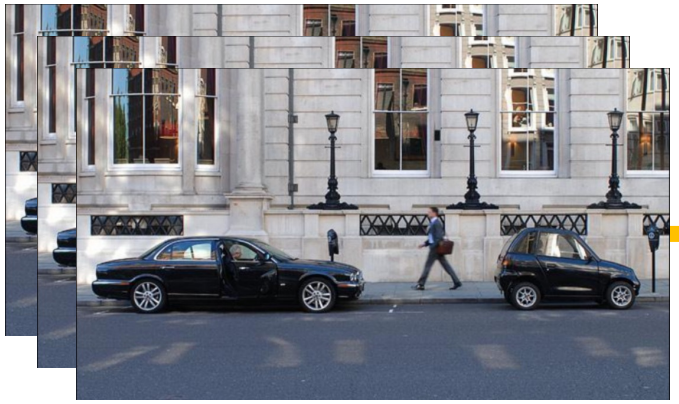
- Object detection
  - Full guide from annotation to detection
- IoT or Embedded systems deployment
  - Model Export
  - Quantization
  - Acceleration
- Other useful resources
  - Webcam
  - Color Order
  - Synthetic Dataset

Object detection

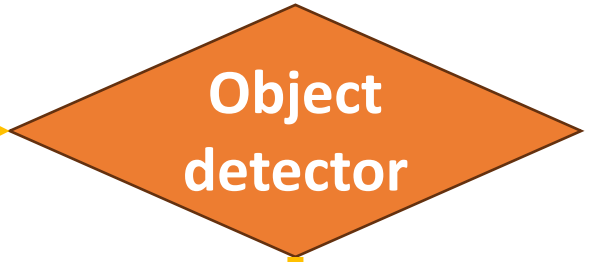
From annotation to detection

Object detection

- Object detection finds the **location and class** of objects in an image.



Input images



Model Outputs



Steps

1. Image collection
2. Annotation
3. Label Conversion
4. Training
5. Evaluation

Before you start.

- Make sure you have the sufficient computational resources to train an object detector.
  - A dedicated GPU (Nvidia) with sufficient VRAM.
    - For example, a NVIDIA GeForce RTX 4070 has 12GB of VRAM
  - Sufficient disk space. (~10GB for small models, ~100GB for larger models.)
  - CPU will be insufficient to train an object detector.

- If you don't have sufficient computational resources, we strongly recommend Google Colab.
  - The free version should be sufficient for most model training tasks.

Image collection

- Taking photos of the items with webcams, phones and cameras.



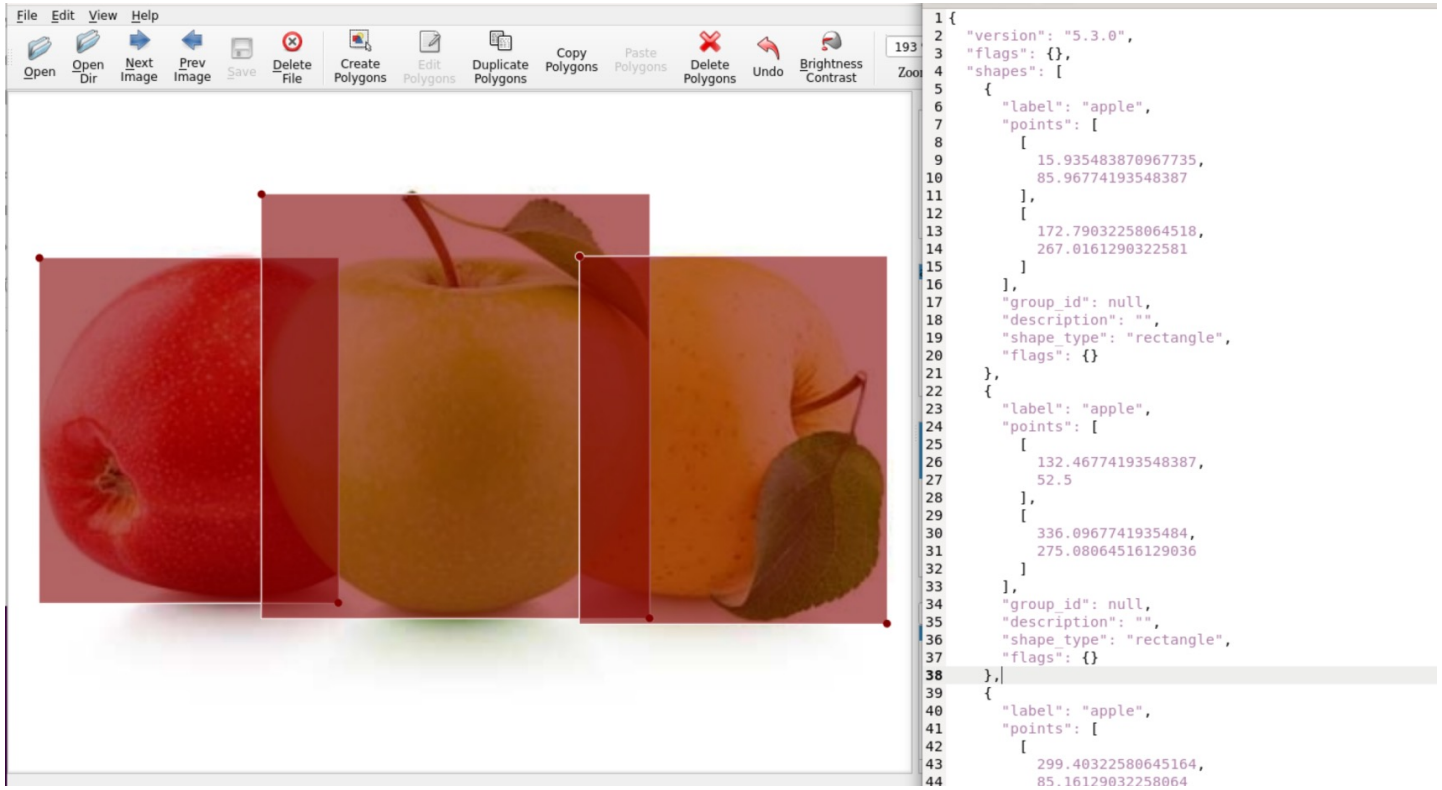
- Typical save format: .jpeg (lossy) and .png (lossless)

Annotation

- The goal is to create a file containing the location and class of the items.



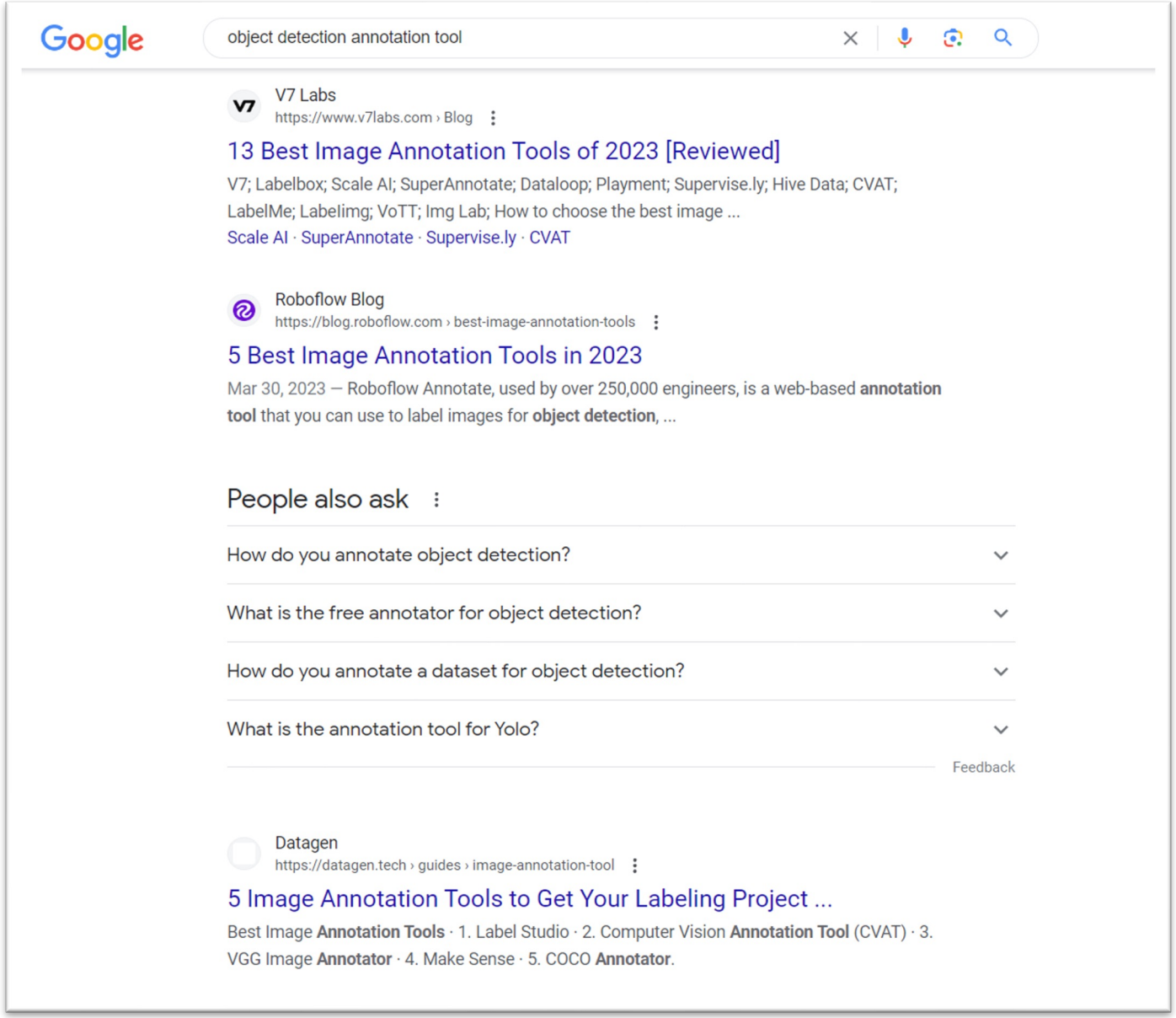
Input images



Locations and classes

Annotation

- There are many annotation tools available on the internet.
- Many are free and open source.
- We will use **labelme** as an example.



Annotation

- Labelme: https://github.com/wkentaro/labelme

README.md

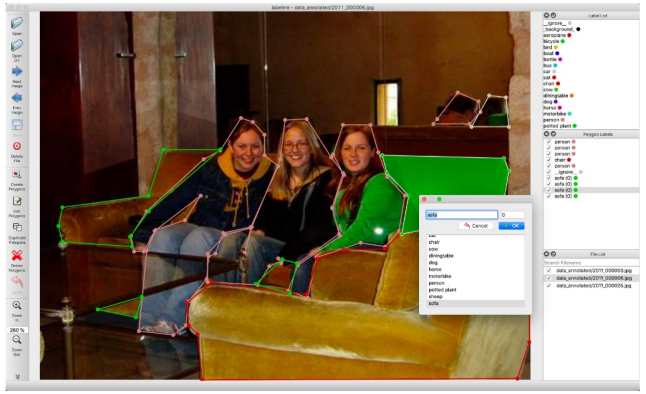


labelme

Image Polygonal Annotation with Python

pypi v5.3.1 | python 3.5 | 3.6 | 3.7 | 3.8 | 3.9 | ci passing

Starter Kit | Installation | Usage | Examples | Community



v5.3.1 Latest

github-actions released this 2 weeks ago · 3 commits to main since t

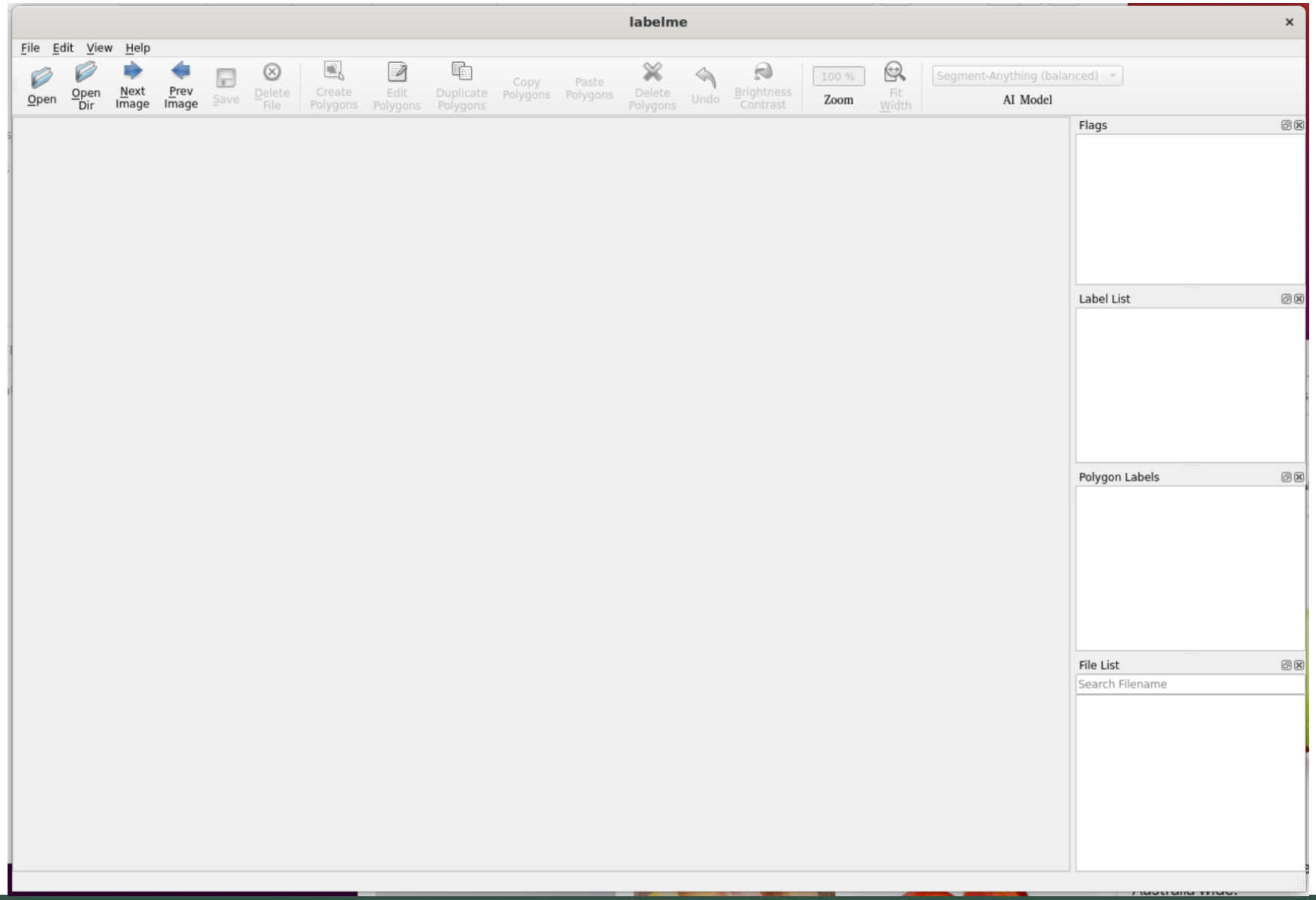
- no changes

Assets 6

- labelme-Linux
- labelme-macOS
- Labelme.dmg
- Labelme.exe
- Source code (zip)
- Source code (tar.gz)

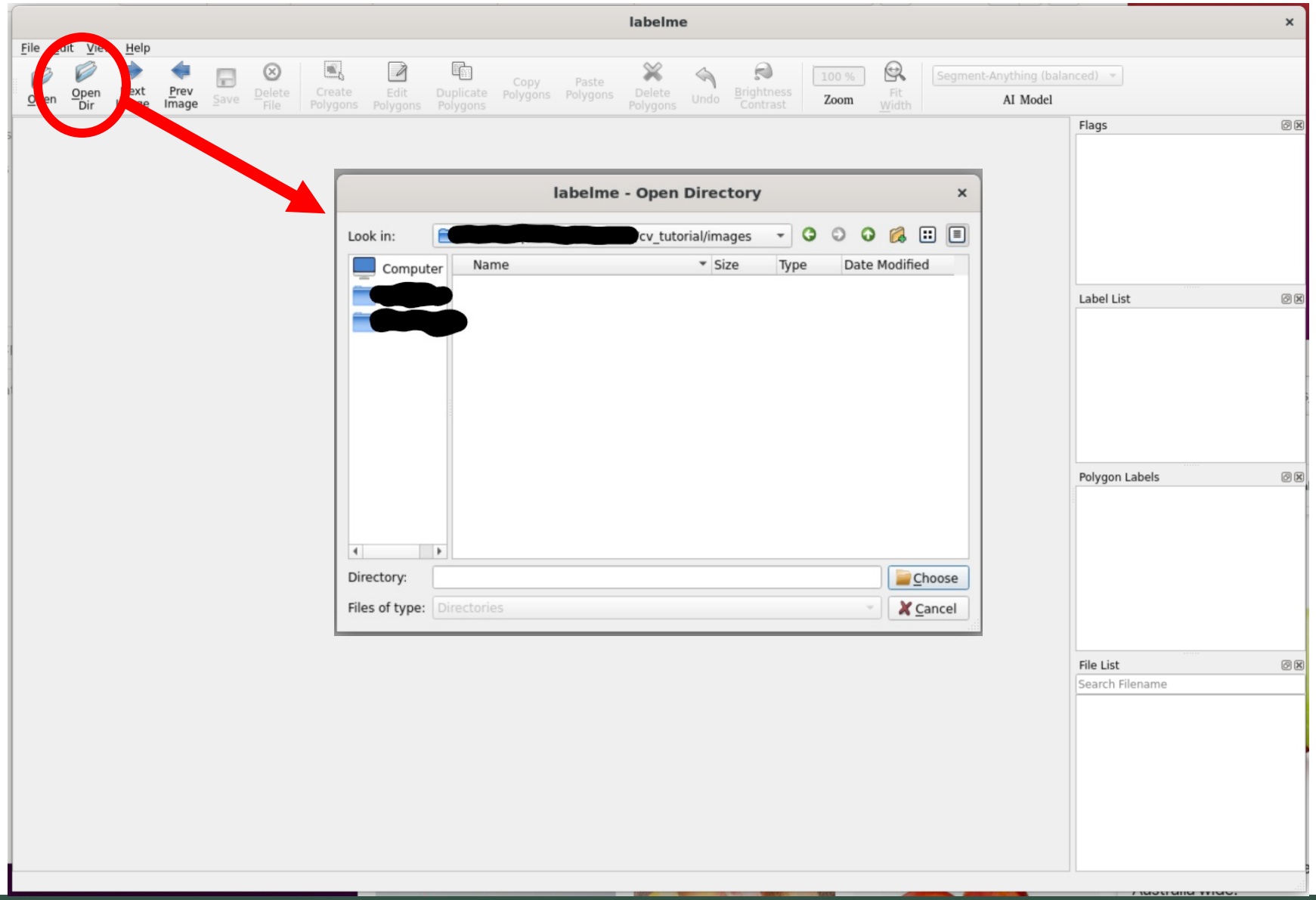
Annotation

The interface of Labelme after opening.



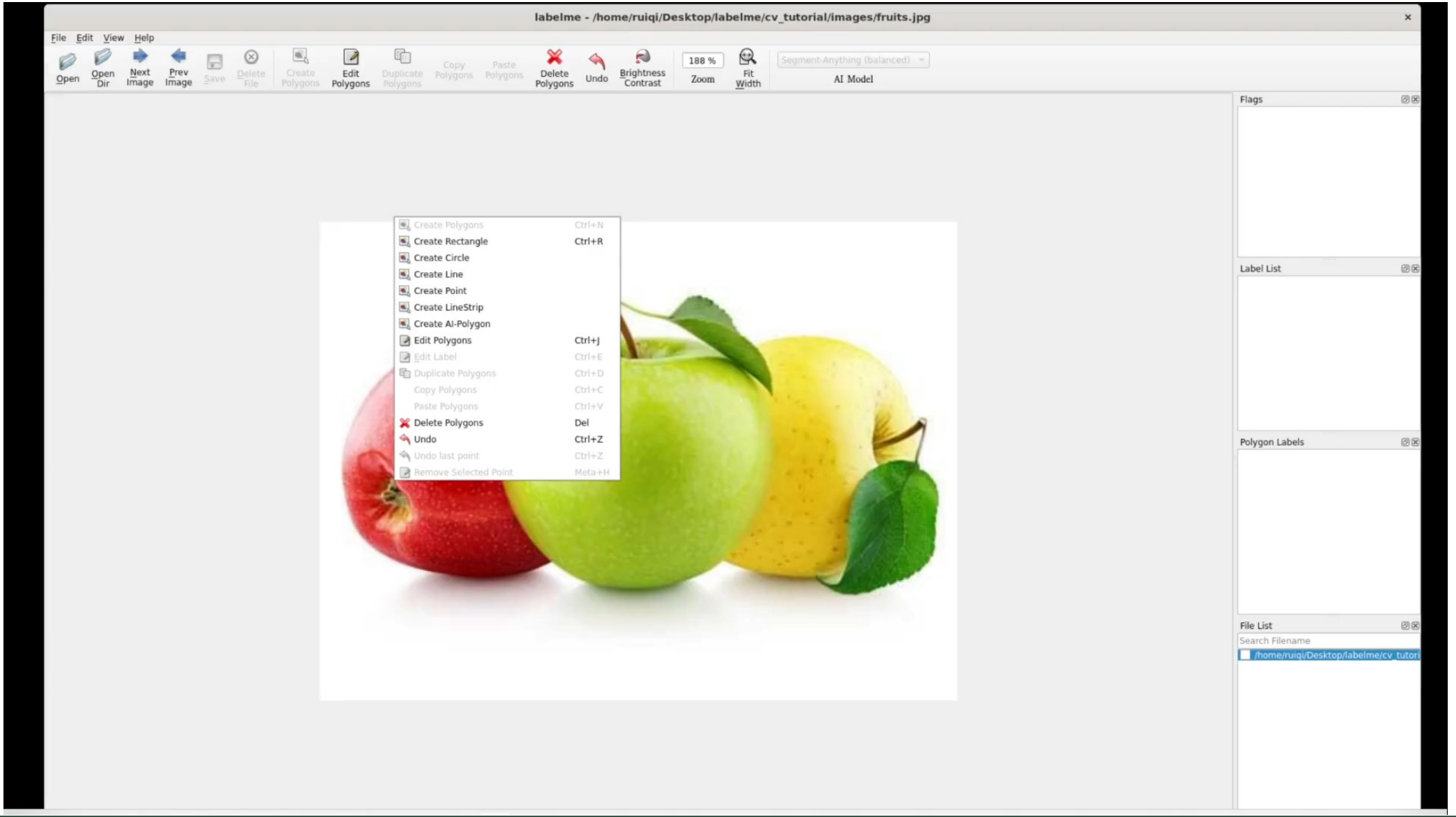
Annotation

The interface of Labelme after opening.



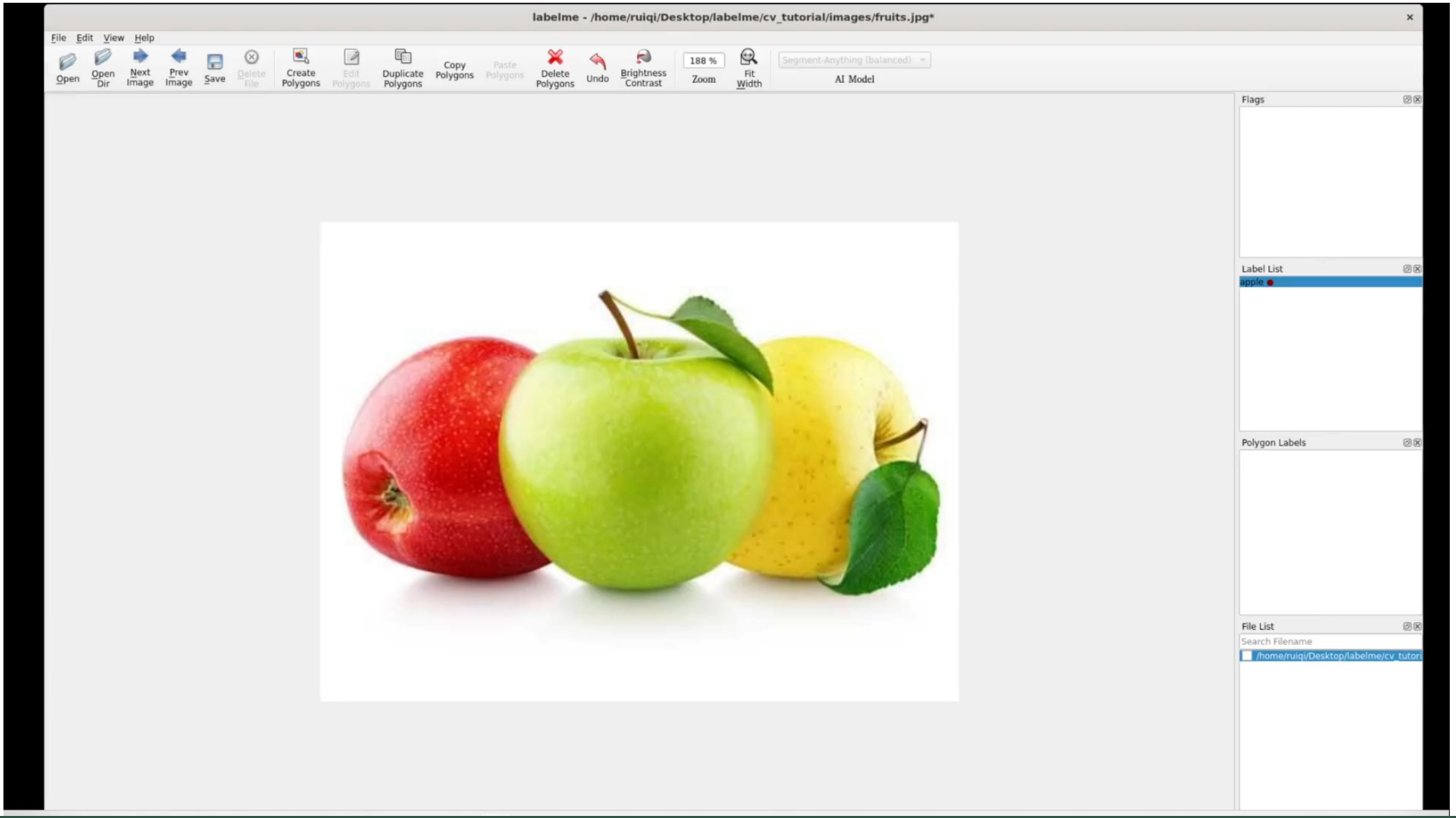
Annotation

Creating polygon annotations.



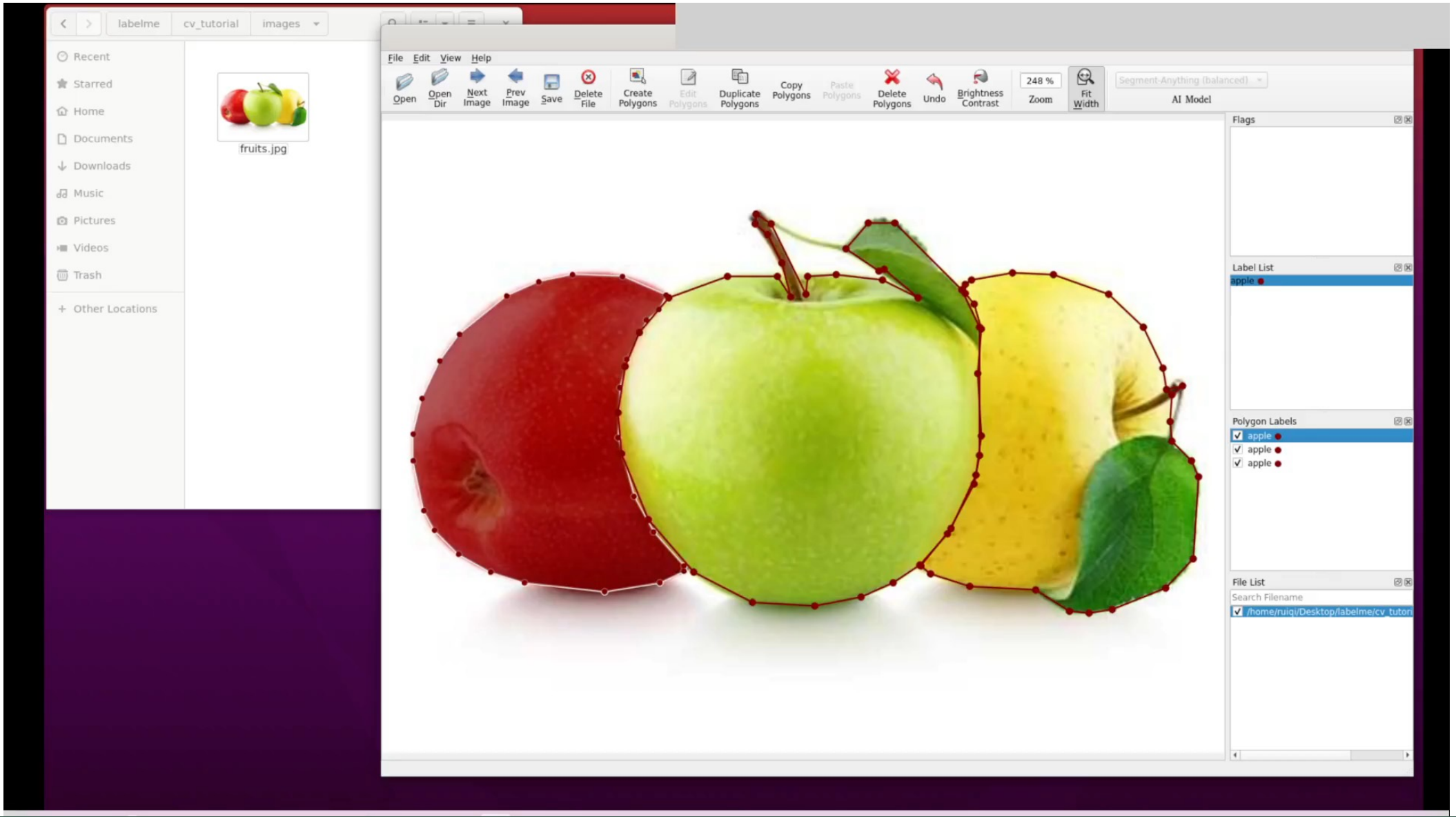
Annotation

Creating AI polygon annotations.



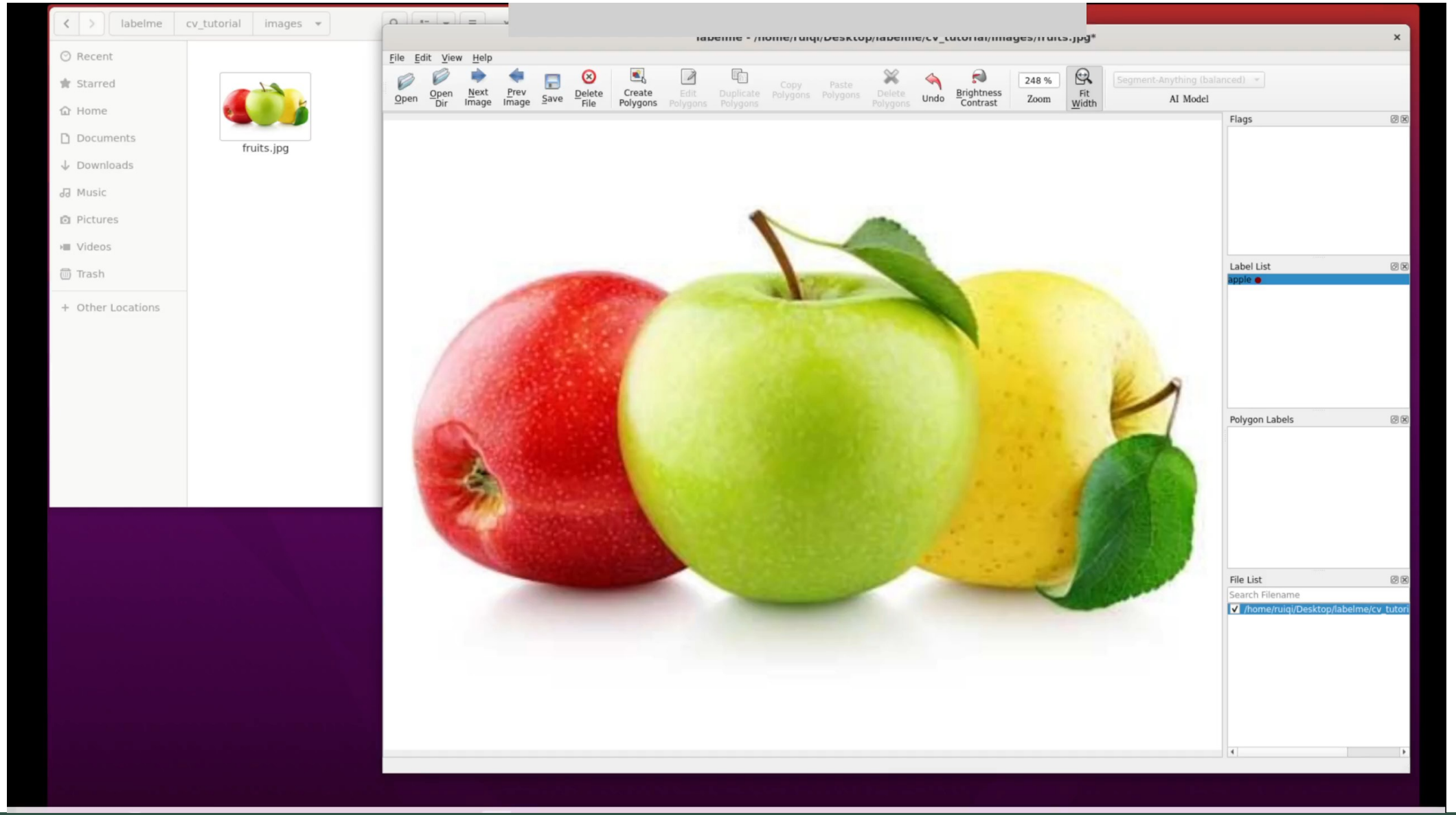
Annotation

Export annotations.



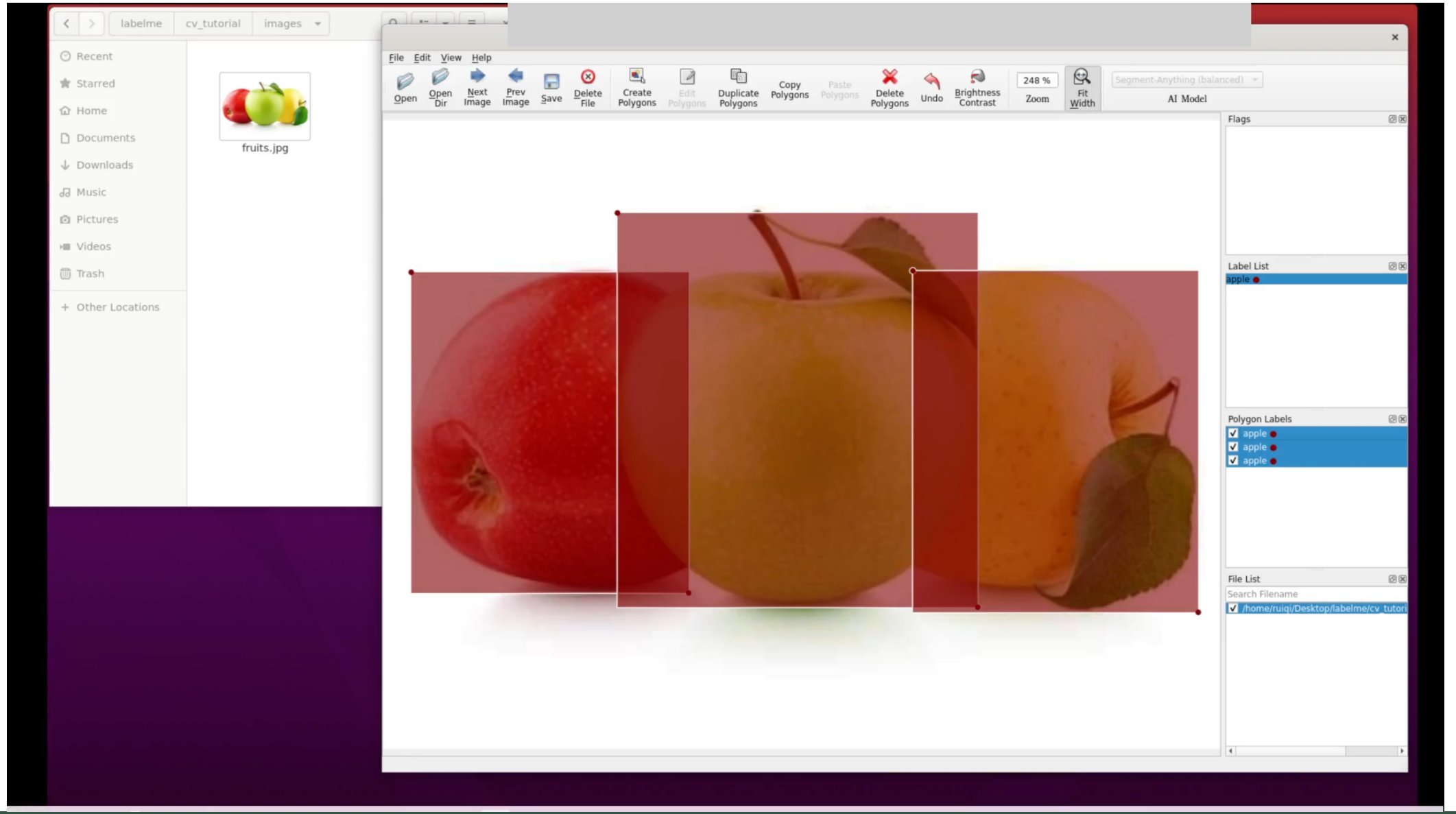
Annotation

Creating rectangle annotations.



Annotation

Export rectangle annotations.



Model Selection

- Select models that can meet your design needs.
- Here are some links for you to find the models you may need:
  - Torch vision pre-trained models: https://pytorch.org/vision/stable/models.html
  - TensorFlow 2 Detection Model Zoo: https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf2_detection_zoo.md
  - YOLO:
    - https://github.com/ultralytics/ultralytics
    - https://github.com/ultralytics/yolov3
  - Other github repos provided by the research papers.

Label Conversion

- After getting your model, you need to check what is **the format of annotation** required by your object detector.
  - The best way to find the required format is to read the model documentation.

COCO (json)

VOC (xml)

YOLO (txt)

- Many annotation tools can export the annotations to your desired annotation format. (Check before you start!)
- More annotation details can be found in:
  - https://www.aitude.com/annotation-converters-for-object-detection/
  - https://towardsdatascience.com/image-data-labelling-and-annotation-everything-you-need-to-know-86ede6c684b1

Label Conversion

- In today's tutorial, we will use **YOLOv5n** as an example. Its online documentation specifies:
  1. the format of the dataset file:

Image & Annotation paths

Defined classes

1.1 Create dataset.yaml

COCO128 is an example small tutorial dataset composed of the first 128 images in COCO train2017. These same 128 images are used for both training and validation to verify our training pipeline is capable of overfitting. data/coco128.yaml, shown below, is the dataset config file that defines 1) the dataset root directory `path` and relative paths to `train` / `val` / `test` image directories (or *.txt files with image paths) and 2) a class `names` dictionary:

```yaml
# Train/val/test sets as 1) dir: path/to/imgs, 2) file: path/to/imgs.txt, or 3) list: [path/to/imgs1, path/to/im
path: ../datasets/coco128  # dataset root dir
train: images/train2017  # train images (relative to 'path') 128 images
val: images/train2017  # val images (relative to 'path') 128 images
test:  # test images (optional)

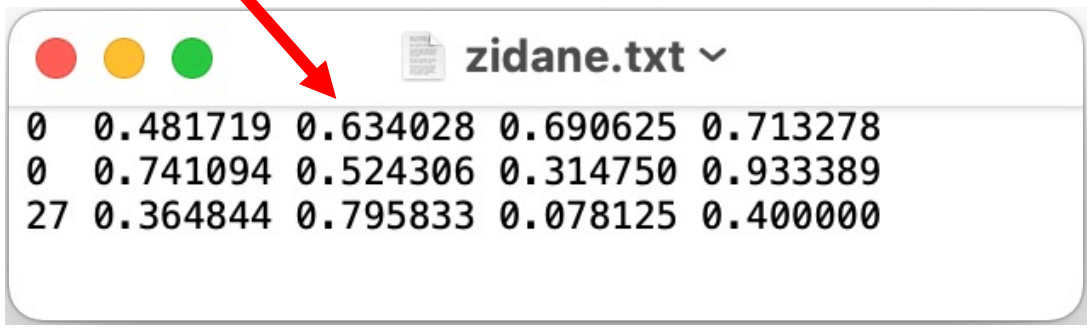
# Classes (80 COCO classes)
names:
  0: person
  1: bicycle
  2: car
  ...
  77: teddy bear
  78: hair drier
  79: toothbrush
```

Source: https://github.com/ultralytics/yolov5/wiki/Train-Custom-Data

Label Conversion

2. The annotation file formats:

- One row per object
- Each row is `class x_center y_center width height` format.



Source: https://github.com/ultralytics/yolov5/wiki/Train-Custom-Data

Label Conversion

3. And where the images and annotations should be placed.

1.3 Organize Directories

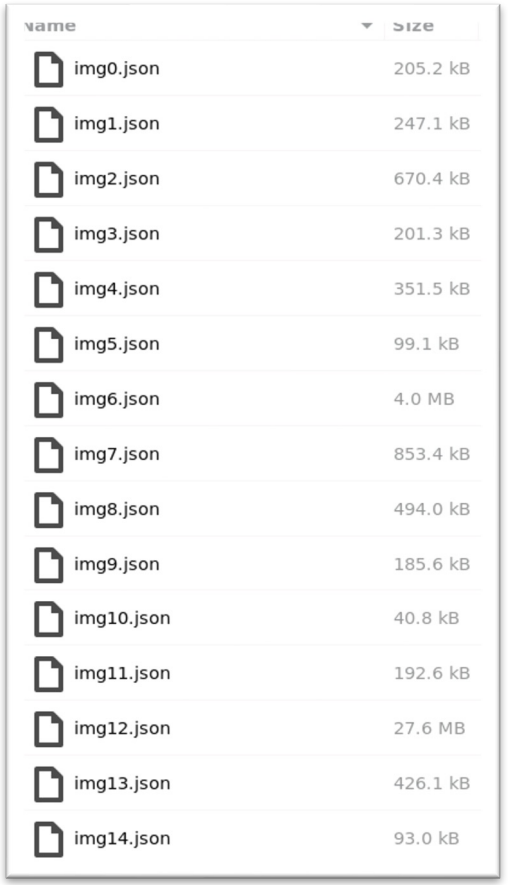
Organize your train and val images and labels according to the example below. YOLOv5 assumes `/coco128` is inside a `/datasets` directory **next to** the `/yolov5` directory. **YOLOv5 locates labels automatically for each image** by replacing the last instance of `/images/` in each image path with `/labels/`. For example:

```
../datasets/coco128/images/im0.jpg  # image
../datasets/coco128/labels/im0.txt  # label
```

Source: https://github.com/ultralytics/yolov5/wiki/Train-Custom-Data

Label Conversion

- Labelme provides conversion code to transform Labelme json format to COCO json format. [1]
- There are also json to YOLO txt conversion code provided by ultralytics [2]



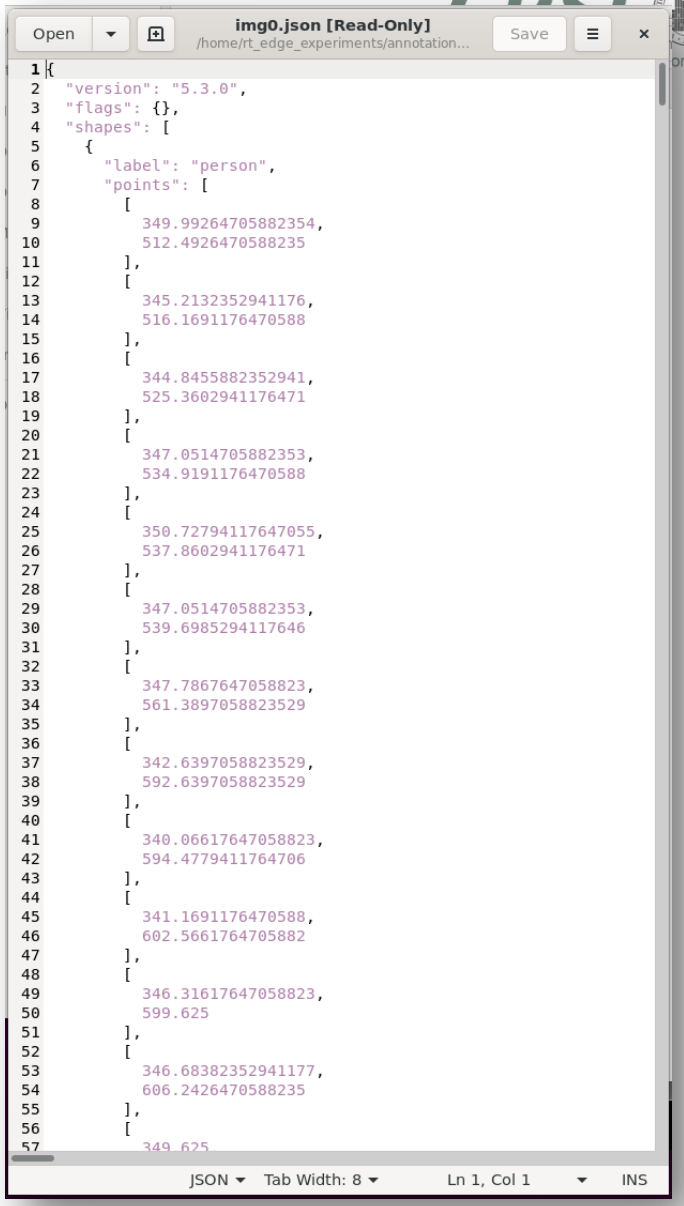
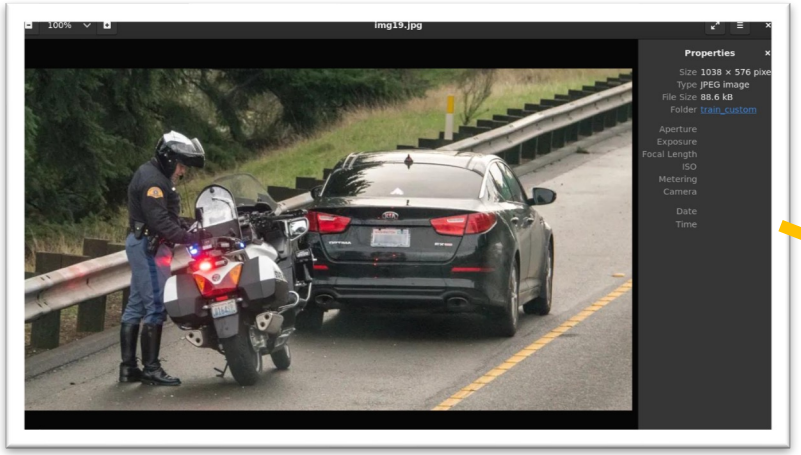


Fig: The file structure and file details from Labelme.

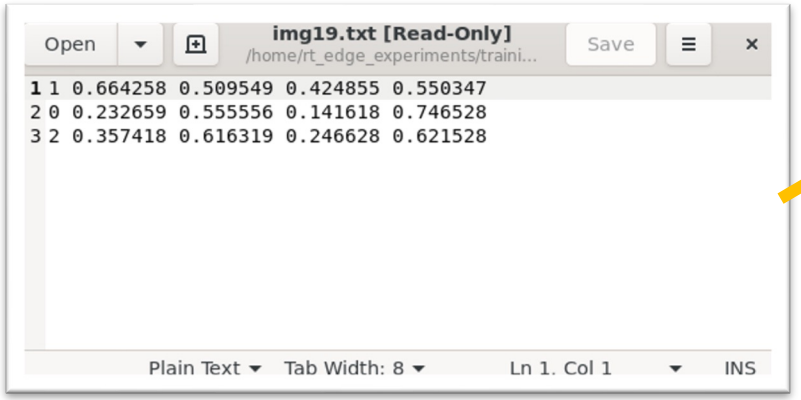
[1] https://github.com/wkentaro/labelme/blob/main/examples/instance_segmentation/labelme2coco.py
[2] https://github.com/ultralytics/JSON2YOLO/tree/master

Label Conversion

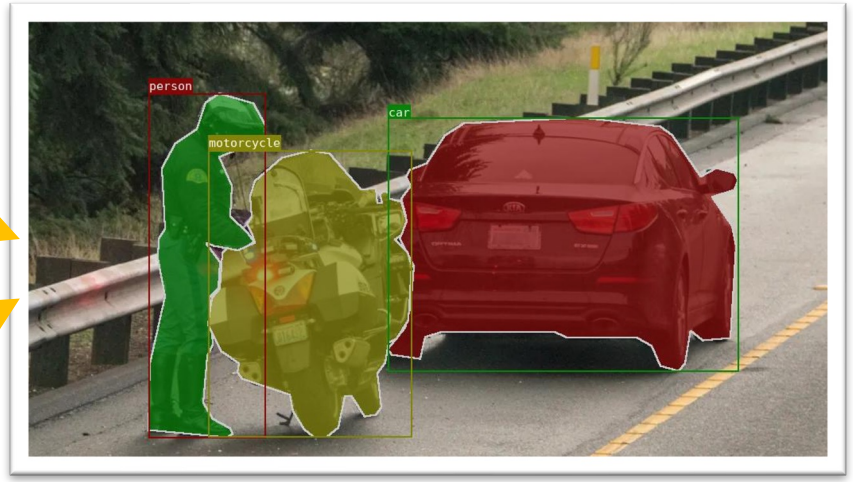
- After conversion, the files to train a YOLO model will look like this:



Input images



Annotations (YOLO txt)



Visualization

Training

- When all the images, annotations and training configs are properly placed, we can train the object detector.

3. Train

Train a YOLOv5s model on COCO128 by specifying dataset, batch-size, image size and either pretrained `--weights yolov5s.pt` (recommended), or randomly initialized `--weights '' --cfg yolov5s.yaml` (not recommended). Pretrained weights are auto-downloaded from the latest YOLOv5 release.

```
# Train YOLOv5s on COCO128 for 3 epochs
$ python train.py --img 640 --batch 16 --epochs 3 --data coco128.yaml --weights yolov5s.pt
```

Source: https://github.com/ultralytics/yolov5/wiki/Train-Custom-Data

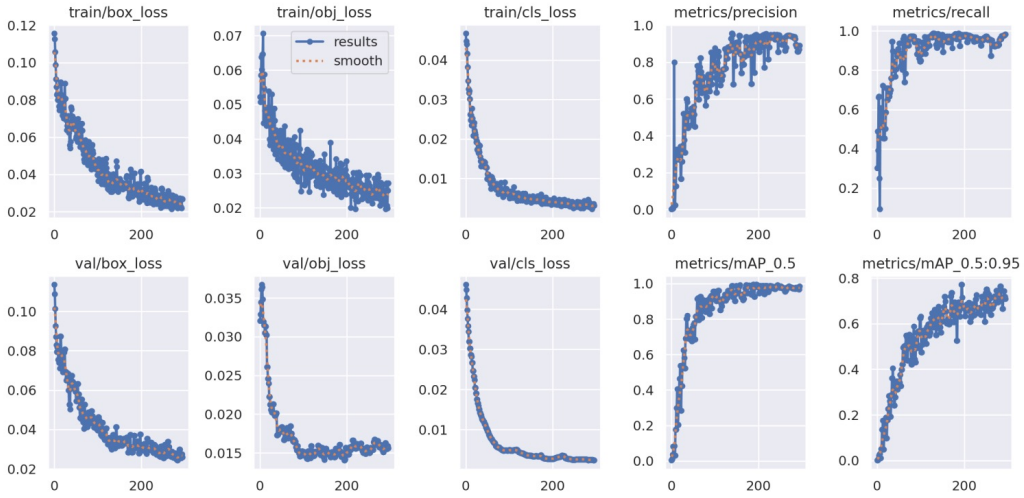
Training

- Train for 1 epoch…

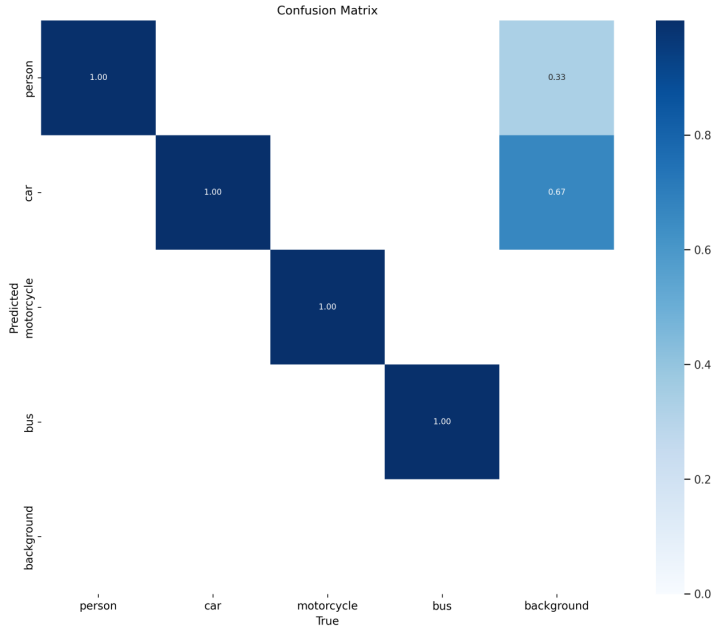


Evaluation

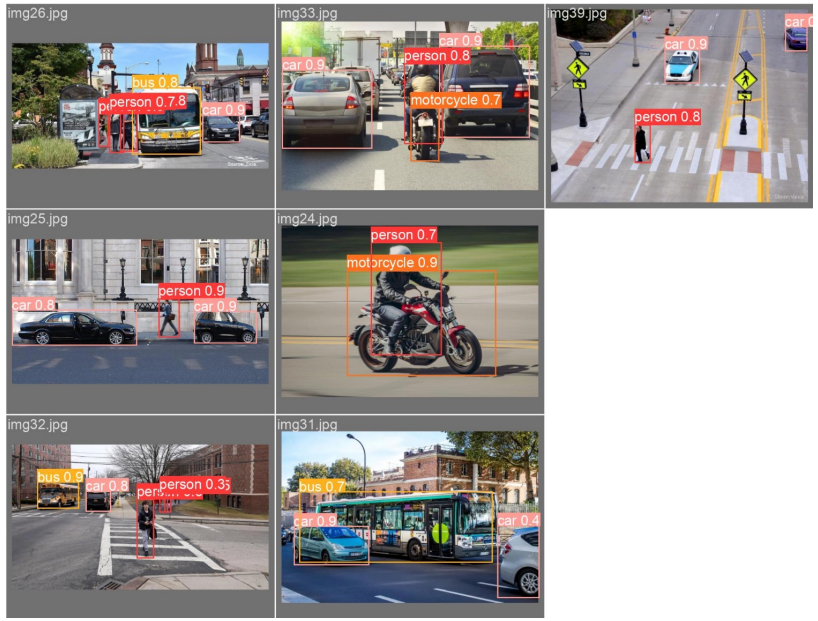
- Many models have evaluation functions to benchmark the training performance:
  - Confusion matrix.
  - Loss curve.
  - Visualizations
  - etc...



Loss curves



Confusion matrix



Visualization

Object Detection Summary

- A tutorial can be found in: https://github.com/brookshu/custom-object-detector/blob/main/tutorial.md
  - Author Brooks Hu is a research intern working with us in the summer of 2023.
- Some relevant codes are also provided in that repo. Including,
  - The conversion codes.
  - The images used in this tutorial.
  - The converted annotations.
  - The code of the yolov5 model.

Deployment on IoT and Embedded Systems

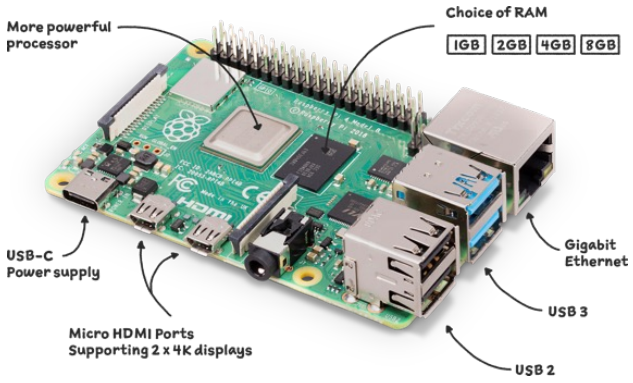
A typical IoT/WSN device

- **Raspberry Pi 4B**

Processor: Broadcom BCM2711, quad-core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz

Memory: 1GB, 2GB, 4GB or 8GB LPDDR4 (depending on model) with on-die ECC

Lower frequency, smaller RAM



- **A gaming computer**

Key Features

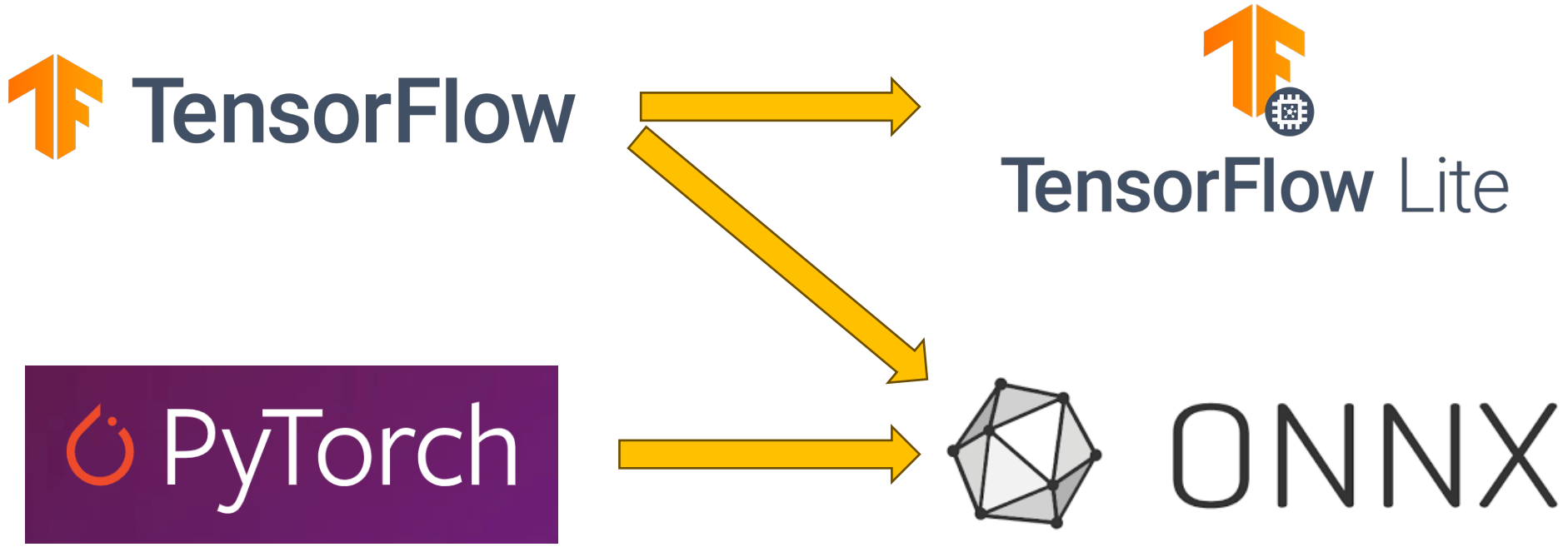
- Intel Core i7 13700KF (2.5GHz)
- G.Skill 32GB DDR5-5600 RAM
- NVIDIA GeForce RTX 4070 Graphics Card
- 2TB SSD
- 2.5GbE LAN, WiFi 6E (802.11ax), Bluetooth 5.1



Source (Top search recommendation of "Gaming PC" from microcenter): https://www.microcenter.com/product/665134/powerspec-g443-gaming-pc

How to make models run efficiently

Step 1: Export the model to a portable format.



- TFLite: https://www.tensorflow.org/lite/models/convert
- ONNX: https://pytorch.org/docs/stable/onnx.html#example-alexnet-from-pytorch-to-onnx

How to make models run efficiently

Step 2: Quantization → ~3x faster.

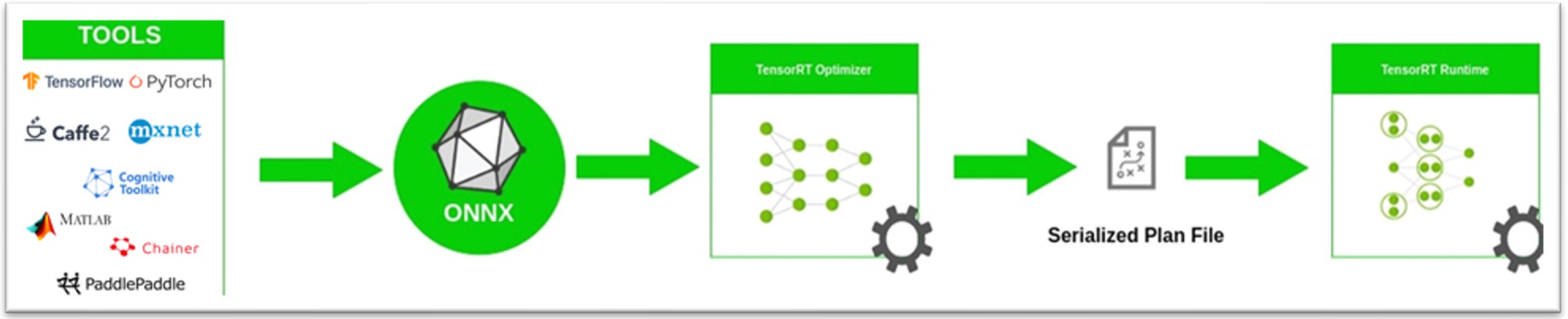
- Resacling from floating point numbers to [0,255]
- Weights are converted FP32 → FP16 → uint8
  - TF: https://www.tensorflow.org/lite/performance/post_training_quantization
  - ONNX: https://onnxruntime.ai/docs/performance/model-optimizations/quantization.html

How to make models run efficiently

Note: First make sure the system works on your computer before attempting any further accelerations!

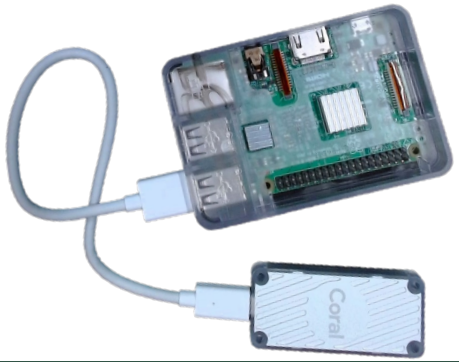
Step 3: Further acceleration

- Further optimization: e.g., TensorRT.



Source: https://developer.nvidia.com/blog/speeding-up-deep-learning-inference-using-tensorflow-onnx-and-tensorrt/

- Additional hardware AI acceleration:



Coral USB



Jetson Nano Developer Kit

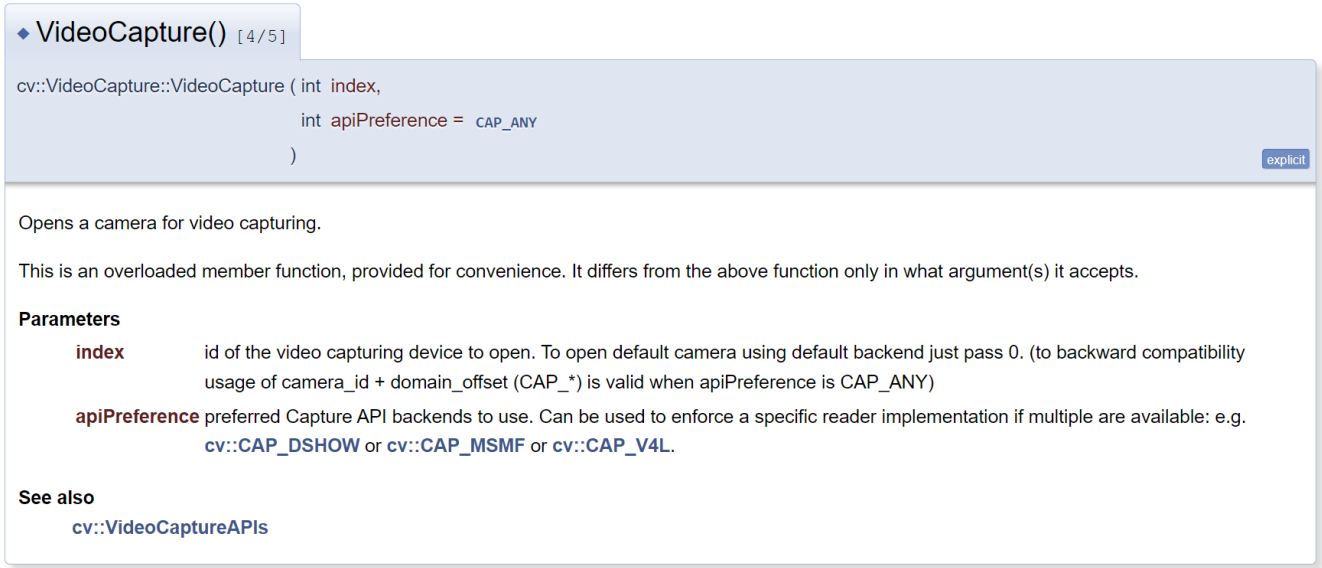
Jetson Boards

Source: https://developer.nvidia.com/embedded/jetson-nano-developer-kit

Other resources

Access webcams with Python

- Before you want to do something related to vision, search if there is a function that do the work in OpenCV.
  - Works on all platforms
  - Install on Raspberry Pi and similar: https://qengineering.eu/install-opencv-on-raspberry-pi.html
- To read frames from a Webcam:
  - OpenCV provides a high-level interface to read the webcam images directly.
  - **Getting Started with Videos (OpenCV documentation):** https://docs.opencv.org/4.x/dd/d43/tutorial_py_video_display.html
  - **Camera test example from Github:** https://github.com/automaticdai/rpi-object-detection/blob/master/src/camera-test/cv_camera_test.py



◆ VideoCapture() [4/5]

```
cv::VideoCapture::VideoCapture ( int  index,
                                 int  apiPreference =  CAP_ANY
                               )                                  explicit
```

Opens a camera for video capturing.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters

index	id of the video capturing device to open. To open default camera using default backend just pass 0. (to backward compatibility usage of camera_id + domain_offset (CAP_*) is valid when apiPreference is CAP_ANY)
apiPreference	preferred Capture API backends to use. Can be used to enforce a specific reader implementation if multiple are available: e.g. **cv::CAP_DSHOW** or **cv::CAP_MSMF** or **cv::CAP_V4L**.

See also

cv::VideoCaptureAPIs

Source: https://docs.opencv.org/4.x/d8/dfe/classcv_1_1VideoCapture.html#aabce0d83aa0da9af802455e8cf5fd181

Access webcams with Python

```python
import numpy as np
import cv2 as cv
cap = cv.VideoCapture(0)
if not cap.isOpened():
    print("Cannot open camera")
    exit()
while True:
    # Capture frame-by-frame
    ret, frame = cap.read()
    # if frame is read correctly ret is True
    if not ret:
        print("Can't receive frame (stream end?). Exiting ...")
        break
    # Our operations on the frame come here
    gray = cv.cvtColor(frame, cv.COLOR_BGR2GRAY)
    # Display the resulting frame
    cv.imshow('frame', gray)
    if cv.waitKey(1) == ord('q'):
    break
# When everything done, release the capture
cap.release()
cv.destroyAllWindows()
```

`cv.VideoCapture(0)`: Opens a camera for video capturing. Usually, the camera you connected will be at index 0.

OpenCV Documentation:
https://docs.opencv.org/4.8.0/d8/dfe/classcv_1_1VideoCapture.html#aabce0d83aa0da9af802455e8cf5fd181

.read(): Grabs, decodes and returns the next video frame.

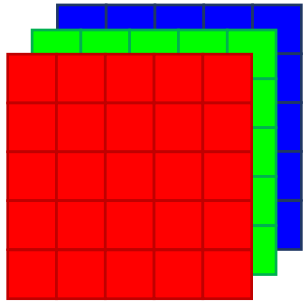
- frame will contain the image array of the webcam.

OpenCV Documentation:
https://docs.opencv.org/4.8.0/d8/dfe/classcv_1_1VideoCapture.html#a473055e77dd7faa4d26d686226b292c1

Code copied from: https://docs.opencv.org/4.x/dd/d43/tutorial_py_video_display.html

Order matters!

- Many models are pre-trained on RGB inputs.



Shape = [3,H,W]

Pillow: PIL.Image.open()

torchvision.io.read_image

RGB by default

If you read your webcam input with OpenCV, pay special attention to the color order!

- Model Confidence: RGB 90% ---> BGR 40%

OpenCV: cv2.imread()

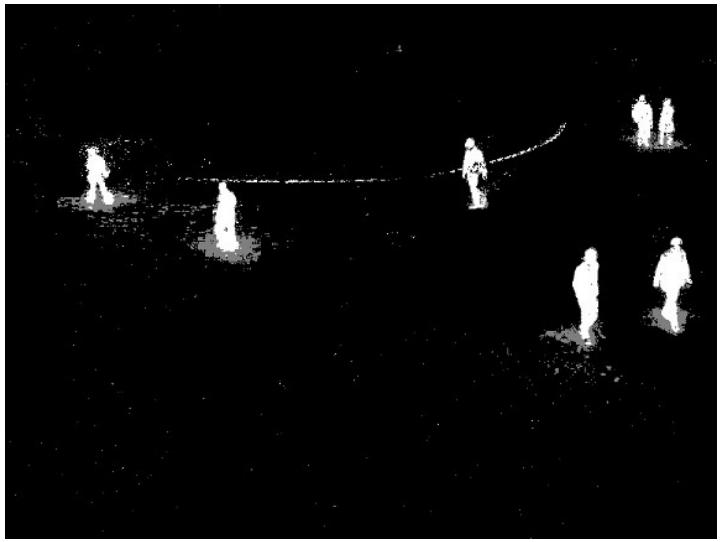
BGR by default

OpenCV Background Subtraction

- Background Subtraction with OpenCV



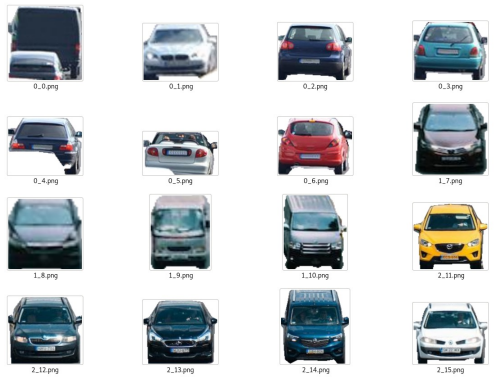
cv.createBackgroundSubtractorMOG2()



Source & OpenCV Doc: https://docs.opencv.org/4.x/d1/dc5/tutorial_background_subtraction.html

Synthetic dataset

- It can be time consuming to collect and annotate many images.
- Generate training images in batches with data synthesis.



Extracted items



Background



Synthetic Data

Hope this tutorial can be helpful to your project!

There are 11 students who haven't sent me the team information. You may raise your hand or stay in the classroom to find possible teammates.

