**YOLOv4 Tiny with the PyTorch framework,**

Below are the steps:

**1. Clone the YOLOv4 PyTorch repository:**

Clone the PyTorch implementation of YOLOv4. Note that YOLOv4 Tiny may be available in specific branches or repositories:

bash

Copy code

git clone https://github.com/WongKinYiu/yolov4-tiny-pytorch.git

cd yolov4-tiny-pytorch

**2. Download pre-trained weights:**

Download the pre-trained weights for YOLOv4 Tiny. You can often find these weights in the repository's releases or README:

bash

Copy code

wget https://github.com/WongKinYiu/yolov4-tiny-pytorch/releases/download/v1.0/yolov4\_tiny.pth

**3. Modify the configuration:**

Adjust the configuration file to suit your needs. This file often contains settings for anchors, input size, and other parameters. Make sure it points to the downloaded weights file.

**4. Inference:**

Use the trained model for inference on an image. Replace input.jpg with your input image:

bash

Copy code

python detect.py --weights yolov4\_tiny.pth --img-size 416 --conf 0.3 --source input.jpg

This command runs the YOLOv4 Tiny model on an image (input.jpg) with a confidence threshold of 0.3.

**5. Custom Training (Optional):**

If you have a custom dataset and want to train YOLOv4 Tiny on it, you'll need to prepare your dataset, modify the configuration accordingly, and run the training script.

bash

Copy code

python train.py --data custom\_data.yaml --cfg cfg/yolov4-tiny-custom.cfg --weights '' --batch-size 8

Adjust the --data, --cfg, and --batch-size parameters based on your dataset and hardware capabilities.

Note:

* Always refer to the specific repository's documentation or README for any additional instructions or updates.
* Ensure you have the necessary dependencies installed, such as PyTorch and torchvision.

This example assumes that you're using the PyTorch implementation provided by the community. Make sure to check the specific repository you are using for any variations or updates to these instructions.

**Yolo**

[Video summary [00:00:00]1](https://edgeservices.bing.com/edgesvc/chat?udsframed=1&form=SHORUN&clientscopes=chat,noheader,udsedgeshop,channelstable,udsdlpconsent,&shellsig=8ceee878a087a7540326760c9ece5531c0b61930&setlang=en-US&lightschemeovr=1#sjevt%7CDiscover.Sydney.SetVideoCurrentTimeEvent%7C0) [- [00:26:05]2](https://edgeservices.bing.com/edgesvc/chat?udsframed=1&form=SHORUN&clientscopes=chat,noheader,udsedgeshop,channelstable,udsdlpconsent,&shellsig=8ceee878a087a7540326760c9ece5531c0b61930&setlang=en-US&lightschemeovr=1#sjevt%7CDiscover.Sydney.SetVideoCurrentTimeEvent%7C1565):

This video is the first part of a series on YOLO object detection models. It explains the main idea and design choices of YOLO version 1, which reframes the object **detection problem as a single-stage regression problem**. It also covers the network architecture, the loss function, and the training process of YOLO v1.

**Highlights**:

* [[00:00:00]3](https://edgeservices.bing.com/edgesvc/chat?udsframed=1&form=SHORUN&clientscopes=chat,noheader,udsedgeshop,channelstable,udsdlpconsent,&shellsig=8ceee878a087a7540326760c9ece5531c0b61930&setlang=en-US&lightschemeovr=1#sjevt%7CDiscover.Sydney.SetVideoCurrentTimeEvent%7C0) **The main idea of YOLO v1**
  + Divides the input image into a grid of cells
  + Each cell predicts **one bounding box** and one objectness score
  + Each cell also predicts a **class probability vector**
  + Combines the predictions to get the final detections
* [[00:04:09]4](https://edgeservices.bing.com/edgesvc/chat?udsframed=1&form=SHORUN&clientscopes=chat,noheader,udsedgeshop,channelstable,udsdlpconsent,&shellsig=8ceee878a087a7540326760c9ece5531c0b61930&setlang=en-US&lightschemeovr=1#sjevt%7CDiscover.Sydney.SetVideoCurrentTimeEvent%7C249) **The network architecture of YOLO v1**
  + Inspired by GoogleNet design
  + Has 24 convolutional layers and 2 fully connected layers
  + Outputs a 7x7x30 tensor of predictions
  + Parses the predictions to get the bounding boxes and class scores
* [[00:10:00]5](https://edgeservices.bing.com/edgesvc/chat?udsframed=1&form=SHORUN&clientscopes=chat,noheader,udsedgeshop,channelstable,udsdlpconsent,&shellsig=8ceee878a087a7540326760c9ece5531c0b61930&setlang=en-US&lightschemeovr=1#sjevt%7CDiscover.Sydney.SetVideoCurrentTimeEvent%7C600) **The loss function of YOLO v1**
  + Sums the losses for each grid cell
  + Consists of three components: box coordinate loss, objectness score loss, and class probability loss
  + Puts more weight on the box coordinate loss
  + Decreases the importance of cells that have no objects
* [[00:15:00]6](https://edgeservices.bing.com/edgesvc/chat?udsframed=1&form=SHORUN&clientscopes=chat,noheader,udsedgeshop,channelstable,udsdlpconsent,&shellsig=8ceee878a087a7540326760c9ece5531c0b61930&setlang=en-US&lightschemeovr=1#sjevt%7CDiscover.Sydney.SetVideoCurrentTimeEvent%7C900) **The training process of YOLO v1**
  + Uses Pascal VOC dataset with 20 classes
  + Pre-trains the network on ImageNet dataset at 224x224 resolution
  + Fine-tunes the network on Pascal VOC dataset at 448x448 resolution
  + Encodes the ground truth labels into relative values with respect to the grid cells

[Video summary [00:26:07]1](https://edgeservices.bing.com/edgesvc/chat?udsframed=1&form=SHORUN&clientscopes=chat,noheader,udsedgeshop,channelstable,udsdlpconsent,&shellsig=8ceee878a087a7540326760c9ece5531c0b61930&setlang=en-US&lightschemeovr=1#sjevt%7CDiscover.Sydney.SetVideoCurrentTimeEvent%7C1567) [- [00:35:22]2](https://edgeservices.bing.com/edgesvc/chat?udsframed=1&form=SHORUN&clientscopes=chat,noheader,udsedgeshop,channelstable,udsdlpconsent,&shellsig=8ceee878a087a7540326760c9ece5531c0b61930&setlang=en-US&lightschemeovr=1#sjevt%7CDiscover.Sydney.SetVideoCurrentTimeEvent%7C2122):

Part 2 of the video talks about the loss function and the performance of YOLO v1, a single-stage object detection model. It covers the components of the loss function, the advantages and limitations of YOLO v1, and the comparison with other models.

**Highlights**:

* [[00:26:07]3](https://edgeservices.bing.com/edgesvc/chat?udsframed=1&form=SHORUN&clientscopes=chat,noheader,udsedgeshop,channelstable,udsdlpconsent,&shellsig=8ceee878a087a7540326760c9ece5531c0b61930&setlang=en-US&lightschemeovr=1#sjevt%7CDiscover.Sydney.SetVideoCurrentTimeEvent%7C1567) **The loss function of YOLO v1**
  + Consists of four terms: bounding box coordinates, objectness score, class probabilities, and no object loss
  + All terms are squared differences between predictions and ground truth
  + Uses different weights for different terms
* [[00:28:32]4](https://edgeservices.bing.com/edgesvc/chat?udsframed=1&form=SHORUN&clientscopes=chat,noheader,udsedgeshop,channelstable,udsdlpconsent,&shellsig=8ceee878a087a7540326760c9ece5531c0b61930&setlang=en-US&lightschemeovr=1#sjevt%7CDiscover.Sydney.SetVideoCurrentTimeEvent%7C1712) **The fast YOLO v1 model**
  + A lighter version of YOLO v1 with only nine layers
  + Achieves higher FPS but lower accuracy
* [[00:30:32]5](https://edgeservices.bing.com/edgesvc/chat?udsframed=1&form=SHORUN&clientscopes=chat,noheader,udsedgeshop,channelstable,udsdlpconsent,&shellsig=8ceee878a087a7540326760c9ece5531c0b61930&setlang=en-US&lightschemeovr=1#sjevt%7CDiscover.Sydney.SetVideoCurrentTimeEvent%7C1832) **The performance of YOLO v1**
  + The fastest model among all detectors
  + Has less false positives than two-stage models
  + Adapts well to other domains
* [[00:32:52]6](https://edgeservices.bing.com/edgesvc/chat?udsframed=1&form=SHORUN&clientscopes=chat,noheader,udsedgeshop,channelstable,udsdlpconsent,&shellsig=8ceee878a087a7540326760c9ece5531c0b61930&setlang=en-US&lightschemeovr=1#sjevt%7CDiscover.Sydney.SetVideoCurrentTimeEvent%7C1972) **The limitations of YOLO v1**
  + Has lower accuracy than two-stage models
  + Can only predict one object per grid cell
  + Has poor localization of bounding boxes

Steps

Preparing datasets for YOLOv4 Tiny involves several steps, including collecting and labeling images, creating configuration files, and organizing the data for training. Here's a step-by-step guide to help you prepare your datasets for YOLOv4 Tiny:

Collect and Organize Your Data:

Gather images relevant to your target object detection task.

Organize the images into separate folders for training and validation.

Labeling:

Annotate the images with bounding boxes around the objects you want to detect using a labeling tool. Popular tools include LabelImg, RectLabel, VGG Image Annotator (VIA), or YOLO Label.

Save the annotations in YOLO format, which includes a text file for each image with the same name as the corresponding image file but with a ".txt" extension.

The YOLO format for each line in the ".txt" file is:

php

Copy code

<class\_index> <center\_x> <center\_y> <width> <height>

<class\_index> is the index of the class (starting from 0).

<center\_x>, <center\_y> are the normalized coordinates of the center of the bounding box.

<width>, <height> are the normalized width and height of the bounding box.

Example:

Copy code

0 0.5 0.5 0.8 0.6

Generate Train and Validation Sets:

Split your dataset into training and validation sets. A common split is around 80% for training and 20% for validation.

Create Classes File:

Create a file containing the names of the classes you want to detect. Each line should contain one class name. Save this file as classes.names.

Download YOLOv4 Tiny Configuration Files:

Download the YOLOv4 Tiny pre-trained weights, configuration file (yolov4-tiny.cfg), and the coco.names file from the official YOLO website (https://github.com/AlexeyAB/darknet).

Modify YOLOv4 Tiny Configuration:

Adjust the configuration file according to your needs. Make changes such as the number of classes, filters, and paths to the training and validation datasets.

Training Configuration:

Set up your training environment with the required dependencies, such as Darknet (the framework for YOLO) and CUDA for GPU acceleration.

Start Training:

Begin training your YOLOv4 Tiny model using the prepared dataset and configuration files. Use the pre-trained weights and fine-tune the model on your specific dataset.

Evaluate and Fine-Tune:

Evaluate the model's performance on the validation set and fine-tune the hyperparameters if necessary.

Testing:

Test your trained YOLOv4 Tiny model on new images to ensure it performs well on unseen data.

Remember to refer to the official YOLO documentation and repositories for the latest updates and recommendations.

# Dataset preparing

# **1. Kaggle download**

# **2. .xml annotation into .txt file format**

# Xml data contains

The information you provided appears to be related to the annotation of an object within an image, likely for the purpose of training an object detection model. Let's break down what each of these fields typically represents in the context of object detection annotations:

Name:

Represents the class or label of the object being annotated. In your example, the object is labeled as "tomato," indicating that the annotated region corresponds to a tomato.

Pose:

Indicates the orientation or pose of the object within the image. In your example, the pose is specified as "Unspecified," suggesting that the annotation does not include information about the specific orientation of the tomato.

Truncated:

Refers to whether the annotated object is partially visible or truncated in the image. A value of 0 typically means the object is not truncated, while a value of 1 might indicate truncation.

Occluded:

Indicates whether the annotated object is partially or fully occluded by other objects or elements in the scene. A value of 0 usually means the object is not occluded, while a value of 1 may indicate occlusion.

Difficult:

Can be used to flag objects that are difficult to detect for various reasons, such as being small, partially obscured, or having ambiguous boundaries. A value of 0 often means the object is not difficult, while a value of 1 may indicate difficulty.

Bndbox (Bounding Box):

Describes the spatial extent of the annotated object using a bounding box. In your example, the bounding box is represented by the <bndbox> element, which includes coordinates such as <xmin>, <ymin>, <xmax>, and <ymax>. These values define the minimum and maximum coordinates of the bounding box in the image.

Here's how the bounding box coordinates are typically interpreted:

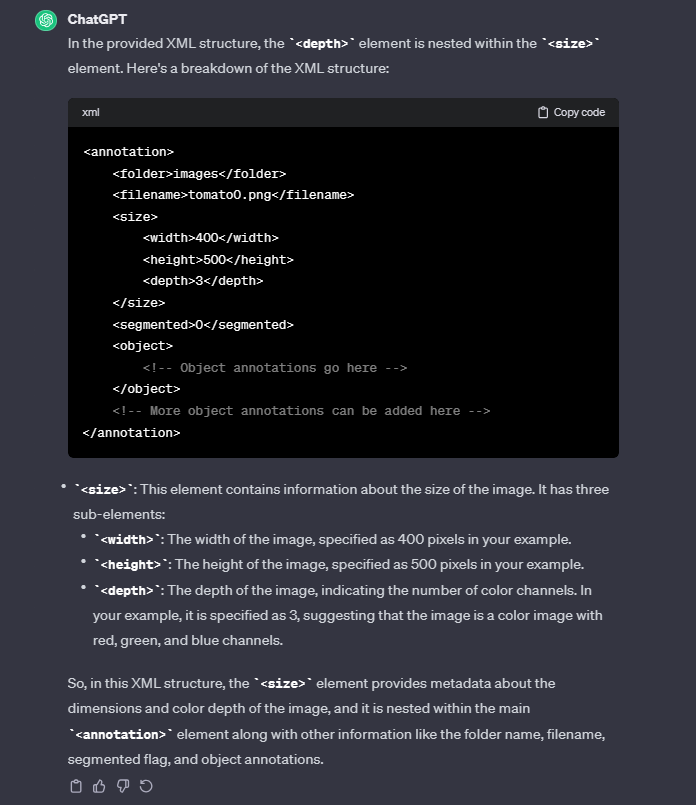
<xmin>: The x-coordinate of the top-left corner of the bounding box.

<ymin>: The y-coordinate of the top-left corner of the bounding box.

<xmax>: The x-coordinate of the bottom-right corner of the bounding box.

<ymax>: The y-coordinate of the bottom-right corner of the bounding box.

These annotations provide crucial information for training object detection models to learn and recognize objects within images. The model uses this labeled data to understand the spatial location, class, and characteristics of objects in the training set, enabling it to make predictions on new, unseen images



In colab

