YOLO versions guide

This is a comprehensive study guide on YOLO (You Only Look Once), tailored for someone who wants to understand the entire ecosystem from the basics to the bleeding edge (as of early 2026).

https://youtu.be/oeXN2fRad_U

Part 1: The Core Concept (What is YOLO?)

YOLO stands for **"You Only Look Once"**.

Before YOLO, object detection models (like R-CNN) were slow because they looked at an image hundreds of times to find potential objects, then classified them.

YOLO changed the game by framing object detection as a **single regression problem**. It looks at the entire image **once** (in one forward pass of the neural network) and predicts bounding boxes and class probabilities simultaneously. This makes it incredibly fast and suitable for real-time applications.

Part 2: The Timeline of Versions

The version history is messy because different authors took over the project. Here is the evolution you need to know:

The "Legacy" Era (History)

- **YOLOv1 - v3 (The Original):** Created by Joseph Redmon. Introduced the grid-based detection system. fast but struggled with small objects.
- **YOLOv4 - v7:** Developed by other researchers (Alexey Bochkovskiy, etc.) after Redmon quit. Focused on "Bag of Freebies" (optimizations that improve accuracy without slowing down inference).
  - *Note for study:* You generally don't need to use these anymore unless you are maintaining old legacy systems.

The "Modern" Era (Ultralytics)

This is where the industry is today. These models are built on **PyTorch**, are easy to use, and support multiple tasks (segmentation, pose, etc.).

1. YOLOv8 (The Industrial Standard)

- **Released:** Early 2023.
- **Significance:** This was the first version to fully unify all tasks (Detection, Segmentation, Classification, Pose) into one easy-to-use API.
- **Key Tech:**
  - **Anchor-Free:** It stopped using "anchor boxes" (pre-defined shapes) and predicts the center of an object directly. This made it more flexible.
  - **Mosaic Augmentation:** It trains by stitching 4 images together, forcing the model to learn context and small objects better.

2. YOLOv11 (The Refined Powerhouse)

- **Released:** Late 2024.
- **Significance:** An architectural refinement of v8. It didn't reinvent the wheel but made the engine much more efficient.
- **Key Tech:**
  - **C3k2 Blocks:** A new type of building block in the neural network that extracts features faster than v8.
  - **Parameter Efficiency:** It is roughly 20-30% faster and lighter than v8 for the same accuracy. If you are building for mobile phones, you use v11 over v8.

3. YOLOv26 (The Bleeding Edge - 2026)

- **Status:** The current State-of-the-Art (as of 2026).
- **Significance:** A massive leap forward because it is **End-to-End**.
- **Key Tech (Study this carefully):**
  - **NMS-Free:** Previous YOLOs predicted multiple boxes for one dog and used a filter called "Non-Maximum Suppression" (NMS) to delete duplicates. YOLOv26 is smart enough to predict **exactly one box** per object. This removes a huge computational bottleneck.
  - **MuSGD Optimizer:** Uses a new training optimizer (inspired by LLMs) to learn faster.
  - **Latency:** Because it removed NMS, it is significantly faster on CPUs and Edge devices (like Raspberry Pi).

Part 3: Decoding the Suffixes (The Tasks) -> for 8, 11, 26

When you see a model name like `yolov8n-seg.pt`, it has three parts:

1. `yolov8`: The Version.
2. `n`: The **Scale** (Nano, Small, Medium, Large, X-Large).
3. `-seg`: The **Task**.

Here is "every thing" about what those tasks mean and how they work:

1. No Suffix (e.g., `yolov8n.pt`) -> Object Detection

- **What it does:** Draws a box around the object.
- **Output:** `[x, y, width, height, class, confidence]`
- **Use Case:** Counting cars, finding faces, spotting defects.

2. `-seg` (e.g., `yolov8n-seg.pt`) -> Instance Segmentation

- **What it does:** Draws a box **AND** wraps a tight polygon mask around the exact pixels of the object.
- **How it works (The Tech):**
  - It has a "Protomask" head. It generates a set of "prototype masks" (fuzzy shapes) and learns to combine them linearly to create the final sharp mask for that specific dog or cat.
- **Use Case:** Background removal, medical imaging (exact tumor shape), self-driving cars (drivable road area).

3. `-pose` (e.g., `yolov8n-pose.pt`) -> Pose Estimation

- **What it does:** Finds "Keypoints" (skeleton joints) on a human or animal.
- **How it works:**
  - Standard YOLO predicts a box.
  - `-pose` adds a head that predicts 17 specific points (Nose, Left Eye, Right Shoulder, etc.) relative to that box center.
- **Use Case:** Gym trackers (checking squat form), analyzing sports movement, fall detection in nursing homes.

4. `-cls` (e.g., `yolov8n-cls.pt`) -> Image Classification

- **What it does:** Tells you *what* is in the image, but **NOT** where it is. No boxes.
- **How it works:** It uses the same "Backbone" (feature extractor) as the detector but cuts off the detection head and replaces it with a simple classification layer.
- **Use Case:** Sorting products on a conveyor belt where the item is always centered.

5. `-obb` (e.g., `yolov8n-obb.pt`) -> Oriented Bounding Box

- **What it does:** Draws a box that can **rotate**.
- **Why we need it:** Standard boxes are axis-aligned (straight up/down). If you detect a ship in a satellite photo, a straight box will include lots of water. An OBB fits the ship perfectly at an angle.
- **Use Case:** Satellite imagery, reading text on rotated documents, package handling.

Part 4: Architecture Deep Dive (For your study)

If you want to master YOLO, you must understand its anatomy. All modern YOLOs (v8, v11, v26) share this 3-part structure:

1. The Backbone (The Eye)

- **Role:** Takes the image and extracts "features" (edges, textures, patterns).
- **Tech:** Usually a **CSP (Cross Stage Partial)** Network. It splits the data flow into two paths and merges them later. This makes the gradient flow richer and the model learns better without getting heavier.

2. The Neck (The Brain)

- **Role:** Mixes features from different scales.
- **Why?** Small objects (like a distant dog) appear in high-resolution detection layers. Big objects (like a dog close up) appear in low-resolution layers. The "Neck" (typically a **PANet** or **BiFPN**) ensures the model can spot the dog regardless of whether it's tiny or huge.

3. The Head (The Hand)

- **Role:** Makes the final prediction.
- **Tech:**
  - **Decoupled Head:** In older versions, one layer predicted "Box" and "Class" together. Modern YOLOs separate them. One branch asks "Where is it?" and another parallel branch asks "What is it?". This improves accuracy significantly.

Summary Table for Quick Reference

Version	Best For	Architecture Note	Standard
YOLOv8	General Purpose / Legacy Support	Anchor-Free, Decoupled Head	The reliable workhorse.
YOLOv11	Mobile / Efficiency	C3k2 Blocks, parameter efficient	Best for phones/Edge.
YOLOv26	**Current Best (SOTA)**	**NMS-Free**, End-to-End	Fastest inference, no post-processing.

Timelilne

Phase 1: The Darknet Era (Legacy)

These models were written in C (Darknet framework). They are the foundation of everything.

Version	Release Date	Authors	The "One Big Thing"
YOLOv1	**May 2016** (CVPR)	Joseph Redmon	**The Birth.** First object detector to look at an image *once* (single stage) rather than scanning it thousands of times. Fast but imprecise.
YOLOv2	**Dec 2016**	Joseph Redmon	**Anchor Boxes.** Introduced "anchors" (pre-defined shapes) to help the model predict box sizes better. Also called "YOLO9000" because it could detect 9000 categories.
YOLOv3	**April 2018**	Joseph Redmon	**Multi-Scale Detection.** Added 3 different detection layers (Small, Medium, Large) to finally fix the issue of detecting small objects. **This is the most famous legacy model.**
YOLOv4	**April 2020**	Alexey Bochkovskiy	**Bag of Freebies.** Redmon quit computer vision. Alexey took over and optimized the architecture with "Mish" activation and advanced data augmentation (Mosaic).

Phase 2: The PyTorch Era (Modern)

The industry shifted to PyTorch for ease of use. These are the models you will likely use in production.

Version	Release Date	Author / Org	The "One Big Thing"
YOLOv5	**June 2020**	Ultralytics (Glenn Jocher)	**Usability.** The first "User Friendly" YOLO. It wasn't purely a research paper; it was a product designed to be easy to install (`pip install`) and train on custom data.
YOLOv6	**June 2022**	Meituan (China)	**Industry Focus.** Optimized specifically for hardware (GPUs) rather than just theoretical accuracy. Introduced "RepVGG" blocks (re-parameterization).
YOLOv7	**July 2022**	Wang & Bochkovskiy	**Trainable Bag-of-Freebies.** The "sequel" to v4. It pushed the limits of how many optimizations you could cram into a model without slowing it down.
YOLOv8	**Jan 2023**	Ultralytics	**Unified Framework.** The first to support *all* tasks (Detection, Segmentation, Pose, Classification) in one simple API. **Anchor-Free** detection became standard here.
YOLOv9	**Feb 2024**	Wang et al.	**PGI (Programmable Gradient Info).** A research-heavy model that focused on keeping data "lossless" as it moved through the deep network layers.
YOLOv10	**May 2024**	Tsinghua Univ.	**NMS-Free (Early attempt).** The first real attempt to remove the NMS filter, though it was slightly experimental and less stable than the industrial versions.

Phase 3: The Current Era (Bleeding Edge)

As of early 2026, these are the standards.

Version	Release Date	Author / Org	The "One Big Thing"
YOLO11	**Sept 2024**	Ultralytics	**Efficiency.** (Note: No "v"). They refined the v8 architecture with **C3k2 blocks** to make it faster on mobile devices. It is the "refined" v8.
YOLO26	**Jan 2026**	Ultralytics	**End-to-End.** (Note: No "v"). The current King. It uses an NMS-Free architecture that is stable enough for production. It is "what you see is what you get"—no post-processing filters needed.

PYTHON

```python
# 1. Update library to get the Jan 2026 definition
!pip install -U ultralytics

# 2. Use the correct name (No 'v', and specify 'n' 's' or 'm')
from ultralytics import YOLO

# 'yolo26s-seg.pt' -> YOLO26 (Small) Segmentation
model = YOLO('yolo26s-seg.pt')
```