YOLO12 Training Handbook

Complete Reference Guide for
Deep Learning Model Training

YOLO12 Family: Nano | Small | Medium | Large | X-Large

Object Detection Training Configuration
Parameter Reference and Best Practices

Author: S.Bertov
Version 1.0.0
26-01-2026

Preface

This handbook serves as a comprehensive reference guide for training YOLO12 family models (Nano, Small, Medium, Large, and X-Large variants). It is designed to be a practical resource that you can consult whenever you need to configure training parameters, understand their effects, or troubleshoot training issues.

How to Use This Handbook

Each parameter in this handbook is documented with the following structure:

1. **Definition** – What the parameter is and what it controls
2. **Purpose** – Why this parameter exists and why it matters
3. **Technical Background** – How the parameter works internally
4. **Value Guidelines** – Recommended values for different scenarios
5. **Dependencies** – How this parameter relates to others
6. **Troubleshooting** – Common problems and solutions

YOLO12 Model Family Overview

Before diving into parameters, understand the model variants you're working with:

Table 1: YOLO12 Model Specifications

Model	Parameters	GFLOPs	Speed	Accuracy	Deployment Target
YOLO12n	∼2.6M	∼6.7	Fastest	Good	Edge devices, Jetson, RPi
YOLO12s	∼9.3M	∼21	Fast	Better	Balanced deployments
YOLO12m	∼20M	∼50	Medium	High	Server, workstations
YOLO12l	∼26M	∼88	Slower	Higher	High-accuracy applications
YOLO12x	∼59M	∼195	Slowest	Highest	Maximum accuracy, cloud

Chapter 1

Training Mode Configuration

1.1 training_mode

1.1.1 Definition

The `training_mode` parameter determines the initialization strategy for model weights at the beginning of training. This is one of the most fundamental decisions you will make when setting up a training pipeline, as it significantly affects training duration, data requirements, and final model performance.

1.1.2 Purpose

Neural networks learn by adjusting millions of numerical weights. Before training begins, these weights must be initialized with some values. The `training_mode` parameter controls whether these initial values are:

- **Random numbers** (raw mode) – The model starts with no prior knowledge
- **Pre-learned values** (pretrained mode) – The model starts with knowledge from a large dataset (typically COCO)

This choice has profound implications for your training process.

1.1.3 Technical Background

Understanding Transfer Learning

When a neural network is trained on a large dataset like COCO (which contains 330,000+ images across 80 object categories), it learns hierarchical features:

1. **Low-level features** (early layers): Edges, corners, textures, color gradients
2. **Mid-level features** (middle layers): Shapes, patterns, object parts
3. **High-level features** (later layers): Complete objects, semantic concepts

The key insight of transfer learning is that low-level and mid-level features are often *universal* – they apply to many different types of images. A model that has learned to detect edges and textures from natural photos can use this knowledge when processing thermal images, satellite imagery, or medical scans.

When Transfer Learning Fails

Transfer learning provides diminishing returns when your target domain differs significantly from the source domain (COCO). COCO consists of natural photographs taken from human eye-level perspective. If your images have fundamentally different characteristics, pretrained features may be irrelevant or even harmful:

Table 1.1: Domain Similarity to COCO Dataset

Image Type	Similarity to COCO	Recommendation
Street cameras (cars, people)	High	Pretrained
Indoor scenes	High	Pretrained
Wildlife photography	Medium-High	Pretrained
Aerial/drone RGB	Low	Raw
Thermal/infrared imagery	Very Low	Raw
Medical imaging (X-ray, MRI)	Very Low	Raw
Satellite imagery	Very Low	Raw
Industrial inspection	Low	Raw
Microscopy	Very Low	Raw

1.1.4 Value Guidelines

Option 1: raw

```
training_mode: raw
```

Use `raw` when:

- Your images are fundamentally different from natural photographs
- You are working with non-visible spectrum imagery (thermal, IR, radar)
- Your perspective is unusual (top-down aerial, microscopic)
- You have sufficient data and computational resources for longer training
- You want maximum control over what the model learns

Note

Raw training typically requires 1.5–2× more epochs than pretrained training to achieve comparable performance, but may ultimately achieve better results on specialized domains.

Option 2: pretrained

```
training_mode: pretrained
```

Use `pretrained` when:

- Your images are similar to natural photographs
- You are detecting common objects (vehicles, people, animals)
- You have limited training data (transfer learning helps with small datasets)

- You need faster training convergence

- Your camera perspective is similar to typical photography

1.1.5 Impact on Other Parameters

The training mode you select affects several other parameters:

Table 1.2: Parameter Adjustments by Training Mode

Parameter	Raw Mode	Pretrained Mode	Reason
epochs	×1.5–2.0	×1.0 (baseline)	Raw needs more iterations
learning_rate	Standard	Can be lower	Pretrained already close to optimal
warmup_epochs	+2 epochs	Standard	Raw weights more unstable
weight_decay	Standard	Can be higher	Prevent overfitting to pretrained

1.1.6 Practical Examples

</> Example

Scenario: Thermal Tank Detection
You are training a model to detect tanks and military vehicles from drone-mounted thermal cameras. The images are grayscale, showing heat signatures rather than visible light, captured from 50-230 meters altitude looking down.
Analysis:

- Thermal imagery is fundamentally different from COCO photos
- Top-down perspective differs from eye-level photography
- Heat signatures have different visual patterns than visible light

Recommendation: raw

```
training_mode: raw
epochs: 1500          # Increased from 800 baseline
warmup_epochs: 5.0    # Increased from 3.0
```

</> Example

Scenario: Parking Lot Vehicle Detection
You are training a model to detect cars in a parking lot using standard RGB security cameras mounted at typical surveillance height.
Analysis:

- Standard RGB imagery similar to COCO
- Vehicle classes exist in COCO dataset
- Perspective is similar to many COCO images

Recommendation: `pretrained`

```
training_mode: pretrained
epochs: 600          # Standard
warmup_epochs: 3.0   # Standard
```

Chapter 2

Class Configuration

2.1 nc (Number of Classes)

2.1.1 Definition

The `nc` parameter specifies the total number of distinct object categories that your model will be trained to detect. This value directly determines the size of the model's output layer and must be consistent across all configuration files.

2.1.2 Purpose

Object detection is fundamentally a classification problem combined with localization. For each potential detection, the model must answer two questions:

1. **Where is the object?** (Bounding box coordinates)
2. **What is the object?** (Class prediction)

The `nc` parameter tells the model how many different answers are possible for the second question. A model trained with `nc: 2` can only distinguish between two types of objects, while `nc: 80` (like COCO) can distinguish between 80 different categories.

2.1.3 Technical Background

Output Layer Architecture

The final detection layer of YOLO produces predictions in the following format for each potential detection:

$$\text{Output} = [x, y, w, h, \text{objectness}, c_1, c_2, ..., c_{nc}] \tag{2.1}$$

Where:

- x, y – Center coordinates of the bounding box
- w, h – Width and height of the bounding box
- objectness – Confidence that an object exists
- $c_1, c_2, ..., c_{nc}$ – Probability for each class

Therefore, increasing `nc` directly increases:

- The number of output neurons
- The number of parameters in the final layer
- The computational cost of the final layer
- The complexity of the classification task

Class Imbalance Considerations

With more classes, you face increased risk of class imbalance – some classes may have significantly more training examples than others. This can cause the model to:

- Perform well on common classes
- Perform poorly on rare classes
- Develop bias toward predicting common classes

2.1.4 Value Guidelines

Minimum Data Requirements

The number of classes directly affects how much data you need for effective training:

Table 2.1: Minimum Training Images per Class

nc	Min Images/Class	Total Min Images	Training Difficulty
1	300–500	300–500	Easy
2	500–800	1,000–1,600	Easy
3–5	800–1,000	2,400–5,000	Moderate
6–10	1,000–1,500	6,000–15,000	Moderate
11–20	1,500–2,000	16,500–40,000	Challenging
21–50	2,000–3,000	42,000–150,000	Difficult
50+	3,000–5,000	150,000+	Very Difficult

Warning

Training with too few images per class leads to overfitting. The model memorizes training examples rather than learning generalizable features. If you don't have enough data, consider reducing the number of classes or collecting more data.

Impact on Training Parameters

More classes generally require adjustments to training parameters:

Table 2.2: Parameter Adjustments by Class Count

Parameter	1–2 classes	3–10 classes	11–30 classes	30+ classes
epochs	Baseline	+10–20%	+20–30%	+30–50%
cls_loss_gain	0.3–0.5	0.5–1.0	1.0–1.5	1.5–2.0
patience	Baseline	+10%	+20%	+30%

2.1.5 Configuration Consistency

Warning

The `nc` value must be **identical** in three locations:

1. Training configuration file (e.g., `train_rgb_n.yaml`)
2. Architecture definition file (e.g., `yolo12n_rgb.yaml`)
3. Dataset configuration file (e.g., `data_rgb.yaml`)

Mismatched values will cause training to crash or produce incorrect results.

2.1.6 Practical Examples

Example

Scenario: Vehicle Detection
You are building a system to detect two types of targets:.

```yaml
# In training config
nc: 2

# In dataset config
nc: 2
names:
  0: class name
  1: class name
```

With only 2 distinct classes, classification is relatively easy. The model primarily focuses on distinguishing the shape and size differences between classes.

Chapter 3

Training Duration

3.1 epochs

3.1.1 Definition

The `epochs` parameter specifies the number of complete passes through your entire training dataset. One epoch means every training image has been seen by the model exactly once.

3.1.2 Purpose

Neural network training is an iterative optimization process. The model cannot learn everything from seeing the data once – it needs multiple exposures to:

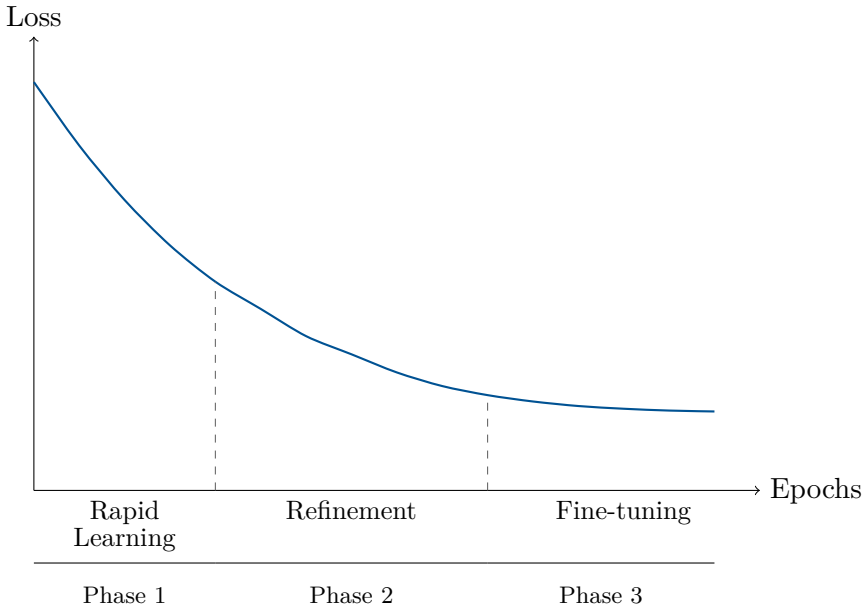
1. **Detect patterns**: Initial epochs identify basic correlations
2. **Refine understanding**: Middle epochs sharpen decision boundaries
3. **Fine-tune details**: Final epochs optimize subtle features

The `epochs` parameter controls how many opportunities the model has to learn from your data.

3.1.3 Technical Background

The Learning Curve

Training progress typically follows a characteristic pattern:



- **Phase 1 (Rapid Learning)**: Loss decreases quickly as the model learns major patterns
- **Phase 2 (Refinement)**: Loss decreases more slowly as the model refines its understanding
- **Phase 3 (Fine-tuning)**: Loss plateaus as the model makes minor adjustments

Relationship with Dataset Size

The number of required epochs is inversely related to dataset size:

$$\text{Total Training Samples} = \text{Dataset Size} \times \text{Epochs} \tag{3.1}$$

A model needs to see approximately the same total number of training samples regardless of dataset size. Therefore:

- **Small dataset (1,000 images)**: Needs many epochs (2000+) to see enough samples
- **Large dataset (100,000 images)**: Needs fewer epochs (200–400) as each epoch provides more learning

3.1.4 Value Guidelines

By Dataset Size

Table 3.1: Recommended Epochs by Dataset Size and Model

Dataset Size	YOLO12n	YOLO12s	YOLO12m	YOLO12l	YOLO12x
$<$ 1,000 images	2000–2500	2500–3000	3000–3500	3500–4000	4000–4500
1,000–3,000 images	1500–2000	1800–2200	2000–2500	2500–3000	3000–3500
3,000–10,000 images	1000–1500	1200–1800	1500–2000	1800–2200	2000–2500
10,000–30,000 images	600–1000	800–1200	1000–1500	1200–1800	1500–2000
30,000–100,000 images	300–600	400–800	500–1000	600–1200	800–1400
$>$ 100,000 images	150–300	200–400	300–500	400–600	500–800

By Training Mode

Table 3.2: Epoch Multiplier by Training Mode

training_mode	Multiplier	Reason
pretrained	1.0× (baseline)	Model starts with useful features
raw	1.5–2.0×	Model must learn everything from scratch

By Training Goal

Table 3.3: Epochs by Training Goal

Goal	Epochs	Use Case
Quick validation	50–100	Verify setup works correctly
Development	300–500	Reasonable results for iteration
Production	800–1500	Good quality for deployment
Maximum quality	1500–2500	Best possible accuracy

3.1.5 Diagnosing Epoch Count Issues

Signs You Need More Epochs

- Loss is still decreasing at the end of training
- mAP metrics are still improving
- Validation performance hasn't plateaued
- Training was interrupted by `patience` early stopping

Signs You Have Too Many Epochs

- Loss has been flat for 100+ epochs
- Training loss decreasing but validation loss increasing (overfitting)
- No mAP improvement for many epochs
- Training time is excessive with no benefit

Tip

Use the `patience` parameter (early stopping) as a safety net. Set epochs higher than you think necessary and let early stopping terminate training when improvement stops. This ensures you don't stop too early while avoiding wasted computation.

Chapter 4

Understanding Batch Size

4.1 batch_size

4.1.1 Definition

The `batch_size` parameter specifies the number of training images that are processed together before the model's weights are updated. This is one of the most important hyperparameters affecting training speed, memory usage, and model quality.

4.1.2 Purpose

In gradient descent optimization, we need to compute how wrong the model's predictions are (the loss) and in which direction to adjust the weights (the gradient). The batch size determines how many examples we use to estimate this gradient:

- **Batch size = 1**: Update weights after every single image (Stochastic Gradient Descent)
- **Batch size = N**: Average gradients over N images before updating (Mini-batch Gradient Descent)
- **Batch size = All**: Use entire dataset for each update (Batch Gradient Descent)

4.1.3 Technical Background

The Gradient Estimation Problem

The true gradient of the loss function would require computing the loss on every possible image. Since this is impossible, we estimate the gradient using a subset (batch) of training images:

$$\text{True Gradient} \approx \frac{1}{N} \sum_{i=1}^{N} \nabla L(x_i) \tag{4.1}$$

Where N is the batch size and $\nabla L(x_i)$ is the gradient for image x_i.

Batch Size Trade-offs

Table 4.1: Batch Size Trade-offs

Aspect	Small Batch (16–32)	Large Batch (128–256)
Gradient Quality	Noisy, high variance	Smooth, low variance
Generalization	Often better (noise acts as regularization)	May be worse (finds sharp minima)
Training Speed	Slow (many updates)	Fast (fewer updates)
Memory Usage	Low	High
Learning Rate	Less sensitive	Very sensitive (requires careful scaling)

The Generalization Debate

Research has shown that smaller batch sizes often lead to better generalization (performance on unseen data). The hypothesis is that the noise in small-batch gradients helps the optimizer:

1. Escape sharp local minima (which generalize poorly)
2. Find flat minima (which generalize better)
3. Act as implicit regularization

However, very small batches make training unstable and slow. The practical sweet spot is typically 32–128 for quality-focused training.

4.1.4 Value Guidelines

By Hardware (VRAM)

The maximum batch size is constrained by your GPU memory. The following tables provide guidance for YOLO12 models:

Table 4.2: YOLO12n Maximum Batch Size by VRAM (640×640 images)

VRAM	Max Batch	Quality	Balanced	Speed
8 GB (RTX 3070)	∼48	24	32	40
10 GB (RTX 3080)	∼64	32	48	56
12 GB (RTX 4080)	∼80	40	56	72
24 GB (RTX 4090)	∼160	80	112	144
32 GB (RTX 5090)	∼200	100	144	176
48 GB (A6000)	∼300	150	208	260
80 GB (A100)	∼480	240	336	420
128 GB (DGX Spark)	∼700	350	500	600

Table 4.3: YOLO12s Maximum Batch Size by VRAM (640×640 images)

VRAM	Max Batch	Quality	Balanced	Speed
8 GB	∼24	12	16	20
12 GB	∼40	20	28	36
24 GB	∼80	40	56	72
32 GB	∼104	52	72	92
48 GB	∼152	76	104	136
80 GB	∼240	120	168	210
128 GB	∼360	180	256	320

Table 4.4: YOLO12m Maximum Batch Size by VRAM (640×640 images)

VRAM	Max Batch	Quality	Balanced	Speed
8 GB	∼12	6	8	10
12 GB	∼20	10	14	18
24 GB	∼40	20	28	36
32 GB	∼52	26	36	46
48 GB	∼76	38	52	68
80 GB	∼120	60	84	108
128 GB	∼180	90	128	160

Table 4.5: YOLO12l Maximum Batch Size by VRAM (640×640 images)

VRAM	Max Batch	Quality	Balanced	Speed
12 GB	∼14	7	10	12
24 GB	∼32	16	22	28
32 GB	∼42	21	30	38
48 GB	∼62	31	44	56
80 GB	∼100	50	70	90
128 GB	∼150	75	105	135

Table 4.6: YOLO12x Maximum Batch Size by VRAM (640×640 images)

VRAM	Max Batch	Quality	Balanced	Speed
24 GB	∼16	8	11	14
32 GB	∼22	11	16	20
48 GB	∼34	17	24	30
80 GB	∼56	28	40	50
128 GB	∼86	43	60	78

Adjustment for Image Size

Larger images require more memory per sample. Adjust batch size accordingly:

$$\text{Adjusted Batch} = \text{Base Batch} \times \frac{640 \times 640}{\text{Width} \times \text{Height}} \tag{4.2}$$

Table 4.7: Batch Size Adjustment by Image Size

Image Size	Pixels	Adjustment Factor	Example (base=160)
640×384	245,760	1.67	267
640×640	409,600	1.00	160
832×480	399,360	1.03	165
1024×576	589,824	0.69	110
1280×720	921,600	0.44	70

By Training Goal

Table 4.8: Batch Size Selection by Training Goal

Goal	Batch %	Example (max=200)	Rationale
Best quality	30–50%	60–100	More updates, better generalization
Balanced	50–70%	100–140	Good compromise
Faster training	70–90%	140–180	Fewer updates, faster epochs
Maximum speed	90–100%	180–200	Fastest possible

4.1.5 Relationship with Learning Rate

Warning

When you change batch size, you should also adjust learning rate. The **Linear Scaling Rule** states:

$$\text{LR}_{\text{new}} = \text{LR}_{\text{base}} \times \frac{\text{Batch}_{\text{new}}}{\text{Batch}_{\text{base}}} \tag{4.3}$$

For example, if baseline is batch=64 with LR=0.01:

- Batch 128 → LR = 0.02
- Batch 192 → LR = 0.03
- Batch 32 → LR = 0.005

4.1.6 Practical Decision Framework

1. Determine your maximum batch size based on VRAM
2. Decide your training goal (quality, balanced, or speed)
3. Select batch size from the appropriate column
4. Calculate the corresponding learning rate
5. Adjust warmup epochs if batch size is large

Example

Scenario: DGX Spark (128GB), YOLO12n, Quality-Focused

1. Maximum batch for YOLO12n on 128GB: ~700
2. Quality-focused: use 30–50% = 210–350
3. Select batch_size = 160 (conservative quality choice)
4. Learning rate: 0.01 × (160/64) = 0.025
5. Warmup: 5 epochs (increased for larger batch)

```
batch_size: 160
val_batch_size: 192
learning_rate: 0.025
warmup_epochs: 5.0
```

4.2 val_batch_size

4.2.1 Definition

The `val_batch_size` parameter specifies the number of images processed together during validation (evaluation) phases.

4.2.2 Purpose

During validation:

- No gradients are computed (no backpropagation)
- No weight updates occur
- Memory requirements are lower than training

Therefore, you can use a larger batch size for validation, which speeds up the evaluation process without affecting model quality.

4.2.3 Value Guidelines

$$\text{val\_batch\_size} = \text{batch\_size} \times 1.2 \text{ to } 1.5 \tag{4.4}$$

Table 4.9: Validation Batch Size Recommendations

Training batch_size	Recommended val_batch_size
32	40–48
64	80–96
96	112–144
128	160–192
160	192–240
192	224–288
256	320–384

```
batch_size: 160
val_batch_size: 192  # 1.2x training batch
```

Chapter 5

Learning Rate Configuration

5.1 learning_rate (lr0)

5.1.1 Definition

The `learning_rate` (also called `lr0` for "learning rate initial") is the most critical hyperparameter in neural network training. It controls the step size for weight updates during optimization.

5.1.2 Purpose

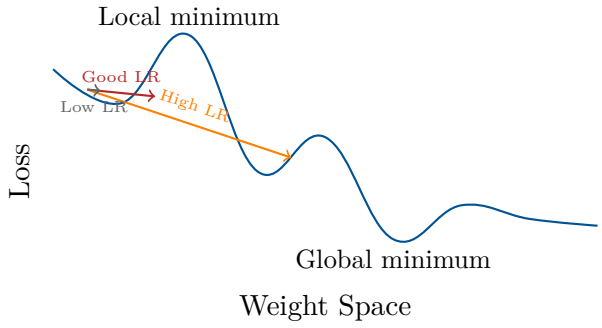
When the model makes a prediction error, we compute how to adjust each weight to reduce the error (the gradient). The learning rate determines how much we actually change each weight:

$$w_{\text{new}} = w_{\text{old}} - \text{learning\_rate} \times \text{gradient} \tag{5.1}$$

5.1.3 Technical Background

The Learning Rate Landscape

Imagine training as navigating a mountainous landscape where altitude represents error:



- **Too high**: Takes huge steps, overshoots valleys, may never converge or explode
- **Too low**: Takes tiny steps, very slow progress, may get stuck in local minima
- **Just right**: Makes steady progress toward the global minimum

Learning Rate Behavior Patterns

Table 5.1: Learning Rate Behavior Patterns

Learning Rate	Loss Behavior	Symptom
Much too high	Explodes to NaN/Inf	Training crashes immediately
Too high	Oscillates wildly	Loss goes up and down unpredictably
Slightly high	Spiky but trending down	Noisy training curve
Optimal	Smooth decrease	Steady improvement
Slightly low	Very slow decrease	Training takes too long
Too low	Barely changes	Loss nearly flat

5.1.4 Value Guidelines

Base Values by Optimizer

Different optimizers have different effective learning rate scales:

Table 5.2: Base Learning Rate by Optimizer (batch size 64)

Optimizer	Base LR	Notes
auto (MuSGD)	0.01	YOLO26 improvement, recommended
SGD	0.01	Classic optimizer
Adam	0.001	Adaptive, 10× lower than SGD
AdamW	0.001	Adam with proper weight decay

The Linear Scaling Rule

When batch size increases, the variance of gradient estimates decreases. To maintain similar training dynamics, learning rate should scale linearly with batch size:

$$\text{LR} = \text{Base LR} \times \frac{\text{Batch Size}}{64} \tag{5.2}$$

Table 5.3: Learning Rate by Batch Size (SGD/MuSGD optimizer)

Batch Size	Calculation	Learning Rate
32	$0.01 \times (32/64)$	0.005
48	$0.01 \times (48/64)$	0.0075
64	$0.01 \times (64/64)$	0.01
96	$0.01 \times (96/64)$	0.015
128	$0.01 \times (128/64)$	0.02
160	$0.01 \times (160/64)$	0.025
192	$0.01 \times (192/64)$	0.03
256	$0.01 \times (256/64)$	0.04 (cap at 0.03–0.04)

Warning

The linear scaling rule has limits. Beyond batch size ~256, the relationship breaks down. Cap learning rate at approximately 0.03–0.04 regardless of batch size.

Adjustments by Model Size

Larger models with more parameters are more sensitive to learning rate:

Table 5.4: Learning Rate Adjustment by Model Size

Model	LR Multiplier	Reason
YOLO12n (2.6M params)	1.0×	Small model, can handle higher LR
YOLO12s (9.3M params)	0.9–1.0×	Medium complexity
YOLO12m (20M params)	0.8–0.9×	Larger model, more sensitive
YOLO12l (26M params)	0.7–0.8×	Large model, needs careful tuning
YOLO12x (59M params)	0.6–0.7×	Largest model, most sensitive

Complete Learning Rate Reference

Table 5.5: Complete Learning Rate Reference Table

Model	Batch	Training Mode	Learning Rate	Warmup
YOLO12n	64	raw	0.01	3
YOLO12n	128	raw	0.02	5
YOLO12n	160	raw	0.02–0.025	5
YOLO12n	192	raw	0.025–0.03	6
YOLO12s	32	raw	0.005–0.008	5
YOLO12s	64	raw	0.01	5
YOLO12s	96	raw	0.015	6
YOLO12s	128	raw	0.018–0.02	8
YOLO12m	16	raw	0.003–0.004	8
YOLO12m	32	raw	0.006–0.008	8
YOLO12m	64	raw	0.01	10
YOLO12m	96	raw	0.012–0.015	12
YOLO12l	16	raw	0.002–0.003	10
YOLO12l	32	raw	0.005–0.006	10
YOLO12l	64	raw	0.008–0.01	12
YOLO12l	80	raw	0.01–0.012	14
YOLO12x	8	raw	0.001–0.002	12
YOLO12x	16	raw	0.003–0.004	12
YOLO12x	32	raw	0.005–0.006	15
YOLO12x	48	raw	0.006–0.008	18

5.1.5 Troubleshooting

Table 5.6: Learning Rate Troubleshooting Guide

Problem	Diagnosis	Solution
Loss becomes NaN	LR much too high	Reduce by 50%, increase warmup
Loss oscillates wildly	LR too high	Reduce by 30%
Loss spikes occasionally	LR slightly high	Reduce by 10–20% or increase warmup
Loss decreases very slowly	LR too low	Increase by 50%
Loss nearly flat	LR much too low	Increase by 100% (double it)

5.2 lrf (Final Learning Rate Ratio)

5.2.1 Definition

The `lrf` parameter specifies the ratio between the final learning rate and the initial learning rate at the end of training.

5.2.2 Purpose

As training progresses:

- Early epochs: Model is far from optimal, needs large steps to make progress
- Late epochs: Model is close to optimal, needs small steps for fine-tuning

The learning rate scheduler gradually reduces LR from the initial value to a final value determined by `lrf`.

5.2.3 Technical Background

$$\text{Final LR} = \text{Initial LR} \times \text{lrf} \tag{5.3}$$

For example, with `learning_rate: 0.02` and `lrf: 0.01`:

$$\text{Final LR} = 0.02 \times 0.01 = 0.0002 \tag{5.4}$$

5.2.4 Value Guidelines

Table 5.7: lrf Values by Training Duration

Training Duration	lrf	Final LR (if lr0=0.02)	Notes
Short ($<$ 500 epochs)	0.1	0.002	Less decay needed
Medium (500–1000)	0.01	0.0002	Standard
Long ($>$ 1000)	0.01	0.0002	Standard works well
Very long ($>$ 2000)	0.001	0.00002	More decay for fine-tuning

```
lrf: 0.01  # Standard value, works for most training scenarios
```

5.3 momentum

5.3.1 Definition

The `momentum` parameter controls gradient smoothing in the optimizer, helping the optimization process overcome local minima and noisy gradients.

5.3.2 Purpose

Without momentum, each weight update is based solely on the current gradient, which can be noisy. Momentum accumulates gradient direction over time, providing:

- **Smoothing**: Reduces oscillation in noisy gradients
- **Acceleration**: Builds up speed in consistent directions
- **Escape**: Helps push through local minima

5.3.3 Technical Background

Momentum maintains a velocity term that accumulates gradient history:

$$v_t = \beta \cdot v_{t-1} + \nabla L(\theta_{t-1}) \tag{5.5}$$
$$\theta_t = \theta_{t-1} - \alpha \cdot v_t \tag{5.6}$$

Where:

- v_t is the velocity (accumulated gradient)
- β is the momentum coefficient
- ∇L is the gradient
- α is the learning rate
- θ are the weights

Physical Analogy

Think of a ball rolling down a hilly surface:

- **No momentum** ($\beta = 0$): Ball stops at every small depression
- **Low momentum** ($\beta = 0.5$): Ball slows at depressions but continues
- **High momentum** ($\beta = 0.99$): Ball rolls over small depressions easily

5.3.4 Value Guidelines

Table 5.8: Momentum Values and Their Effects

momentum	Effective History	Behavior
0.9	∼10 iterations	More responsive, adapts quickly
0.937	∼16 iterations	Balanced (YOLO default)
0.95	∼20 iterations	Smoother, slower to adapt
0.99	∼100 iterations	Very smooth, very slow to change

```
momentum: 0.937  # YOLO default, rarely needs changing
```

Tip

The default value of 0.937 works well in almost all cases. Only adjust if:

- Training is unstable → try 0.95
- Training is too slow → try 0.9

5.4 weight_decay

5.4.1 Definition

The `weight_decay` parameter implements L2 regularization, which penalizes large weight values to prevent overfitting.

5.4.2 Purpose

Neural networks with many parameters can memorize training data instead of learning generalizable patterns. Weight decay discourages this by:

- Adding a penalty for large weights to the loss function
- Encouraging the model to use smaller, distributed weights
- Improving generalization to unseen data

5.4.3 Technical Background

Weight decay adds a term to the loss function:

$$L_{\text{total}} = L_{\text{prediction}} + \frac{\lambda}{2} \sum_{i} w_i^2 \tag{5.7}$$

Where λ is the weight decay coefficient. This causes the weight update to include a decay term:

$$w_{\text{new}} = w_{\text{old}} - \alpha \cdot (\nabla L + \lambda \cdot w_{\text{old}}) \tag{5.8}$$

Effectively, weights are slightly reduced toward zero at each update.

5.4.4 Value Guidelines

Table 5.9: Weight Decay by Dataset Size

Dataset Size	weight_decay	Reasoning
$<$ 1,000 images	0.001	Strong regularization, high overfit risk
1,000–5,000 images	0.0005–0.001	Medium-strong
5,000–20,000 images	0.0005	Standard
20,000–100,000 images	0.0003–0.0005	Light regularization
$>$ 100,000 images	0.0001–0.0003	Minimal, data provides regularization

Diagnosing Weight Decay Issues

Table 5.10: Weight Decay Troubleshooting

Observation	Diagnosis	Action
Val loss ↑, Train loss ↓	Overfitting	Increase weight_decay 2×
Both losses high, not improving	Underfitting	Decrease weight_decay 2×
Training looks normal	Correct value	Keep current

```
weight_decay: 0.0005  # Standard value for most datasets
```

Chapter 6

Warmup Configuration

6.1 warmup_epochs

6.1.1 Definition

The `warmup_epochs` parameter specifies the number of epochs during which the learning rate gradually increases from near-zero to the target value.

6.1.2 Purpose

At the start of training:

- Model weights are randomly initialized
- Gradients are large and unreliable
- The loss landscape is poorly understood

Applying the full learning rate immediately can cause:

- Exploding gradients (loss goes to NaN)
- Unstable training (wild oscillations)
- Getting stuck in bad local minima

Warmup solves these problems by starting gentle and gradually increasing intensity.

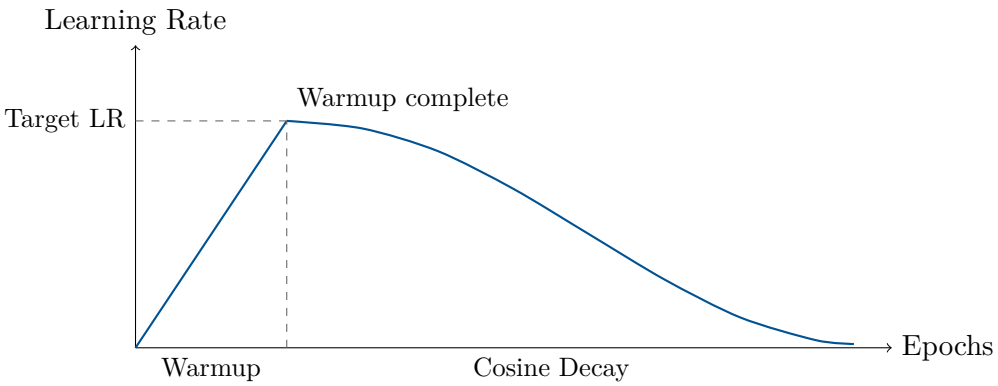
6.1.3 Technical Background

During warmup, the learning rate increases linearly:

$$\text{LR}(e) = \text{LR}_{\text{target}} \times \frac{e}{\text{warmup\_epochs}} \tag{6.1}$$

For $e \leq \text{warmup\_epochs}$, where e is the current epoch.

Visual Representation



6.1.4 Value Guidelines

Base Values by Model

Table 6.1: Base Warmup Epochs by Model Size

Model	Base Warmup	Reason
YOLO12n	3 epochs	Small model, stabilizes quickly
YOLO12s	5 epochs	Medium complexity
YOLO12m	8 epochs	Larger model, needs more stabilization
YOLO12l	10 epochs	Large model, extended stabilization
YOLO12x	12 epochs	Maximum size, longest stabilization

Adjustments

Add epochs based on these factors:

Table 6.2: Warmup Epoch Adjustments

Factor	Condition	Add Epochs
Large batch	batch $>$ 128	+2
Very large batch	batch $>$ 256	+3
High learning rate	LR $>$ 0.02	+2
Very high learning rate	LR $>$ 0.03	+3
Raw training mode	training_mode: raw	+2

Complete Reference

Table 6.3: Warmup Epochs Reference Table

Model	Batch	LR	Mode	warmup_epochs
YOLO12n	64	0.01	raw	3
YOLO12n	128	0.02	raw	5
YOLO12n	160	0.025	raw	5
YOLO12n	192	0.03	raw	6
YOLO12s	32	0.008	raw	5
YOLO12s	64	0.01	raw	5
YOLO12s	96	0.015	raw	6
YOLO12s	128	0.02	raw	8
YOLO12m	16	0.004	raw	8
YOLO12m	32	0.008	raw	8
YOLO12m	64	0.01	raw	10
YOLO12m	96	0.015	raw	12
YOLO12l	8	0.002	raw	10
YOLO12l	16	0.004	raw	10
YOLO12l	32	0.008	raw	12
YOLO12l	48	0.012	raw	14
YOLO12x	4	0.001	raw	12
YOLO12x	8	0.002	raw	12
YOLO12x	16	0.004	raw	15
YOLO12x	24	0.006	raw	18

6.1.5 Diagnosing Warmup Issues

Table 6.4: Warmup Troubleshooting

Symptom	Diagnosis	Solution
Loss explodes after warmup	Warmup too short	Increase by 2–3 epochs
Loss spikes at warmup end	Transition too abrupt	Increase warmup slightly
Training very slow at start	Normal for warmup	This is expected behavior
Warmup wastes too much time	Warmup too long	Decrease if stable

Chapter 7

Learning Rate Scheduling

7.1 scheduler

7.1.1 Definition

The `scheduler` parameter defines how the learning rate changes throughout training after the warmup period completes.

7.1.2 Purpose

Optimal learning rate varies during training:

- **Early training**: Higher LR for rapid progress through the loss landscape
- **Mid training**: Moderate LR for steady improvement
- **Late training**: Lower LR for fine-tuning and convergence

The scheduler automates this progression.

7.1.3 Available Schedulers

Cosine Scheduler

```
scheduler: cosine
```

The cosine scheduler follows a smooth cosine curve:

$$\text{LR}(e) = \text{LR}_f + \frac{1}{2}(\text{LR}_0 - \text{LR}_f)\left(1 + \cos\left(\frac{\pi \cdot e}{E}\right)\right) \tag{7.1}$$

Where e is current epoch, E is total epochs, LR_0 is initial LR, and LR_f is final LR.
Characteristics:

- Slow decay at the beginning (maintains high LR longer)
- Fast decay in the middle
- Slow decay at the end (gentle approach to final LR)
- Smooth, continuous transitions

Linear Scheduler

```
scheduler: linear
```

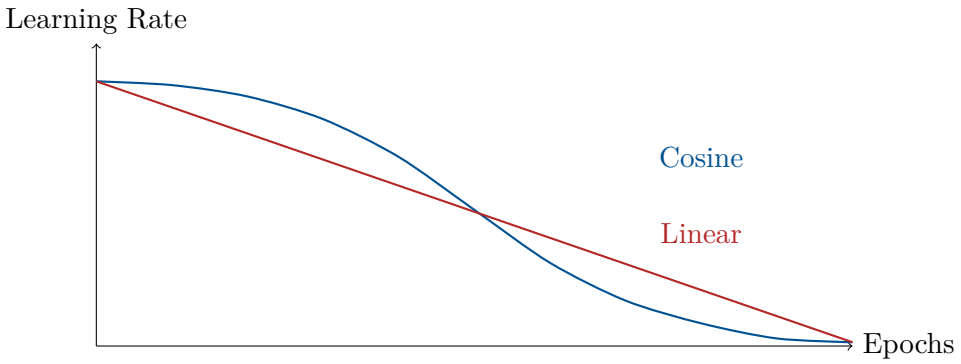
The linear scheduler decreases at a constant rate:

$$\text{LR}(e) = \text{LR}_0 - (\text{LR}_0 - \text{LR}_f) \times \frac{e}{E} \tag{7.2}$$

Characteristics:

- Constant rate of decay
- Simpler, more predictable
- Less time at high LR than cosine

7.1.4 Visual Comparison



7.1.5 Value Guidelines

Table 7.1: Scheduler Selection Guide

Training Duration	Recommended	Reason
< 300 epochs	linear	Quick experiments, simple decay
300–1000 epochs	cosine	Better convergence
> 1000 epochs	cosine	Best for long training
Production training	cosine	Consistently better results

```
scheduler: cosine  # Recommended for production training
```

Chapter 8

Augmentation Overview

8.1 Purpose of Augmentation

Data augmentation artificially increases the diversity of your training data by applying transformations to images. This helps the model:

- **Generalize better**: Learn features that are invariant to irrelevant changes
- **Avoid overfitting**: See more varied examples even with limited data
- **Handle real-world variation**: Become robust to lighting, position, scale changes

8.2 RGB vs Grayscale Configuration

The most important augmentation distinction is between RGB color images and grayscale/thermal images:

Table 8.1: Key Augmentation Differences: RGB vs Grayscale

Parameter	RGB	Grayscale	Reason
hsv_h	0.015	0.0	Grayscale has no hue information
hsv_s	0.7	0.0	Grayscale has no saturation
hsv_v	0.4	0.4	Brightness applies to both
image aspect	16:9	1:1	RGB often widescreen, thermal often square

Chapter 9

Geometric Augmentations

9.1 degrees (Rotation)

9.1.1 Definition

The `degrees` parameter specifies the maximum random rotation angle applied to training images (in both positive and negative directions).

9.1.2 Purpose

Rotation augmentation teaches the model to recognize objects regardless of their orientation in the image. This is valuable when objects can appear at various angles.

9.1.3 Technical Background

When `degrees: 15` is set, each image is randomly rotated by an angle uniformly sampled from $[-15, +15]$.

9.1.4 Value Guidelines

Table 9.1: Rotation Augmentation by Dataset Type

Dataset Type	degrees	Reasoning
Aerial/drone (nadir)	0.0	Objects have consistent orientation (tanks face specific directions, "up" is always north)
Street view cameras	0–10	Slight camera tilt possible, but ground is always down
General object detection	10–45	Objects can appear at various angles
Rotation-invariant objects	90–180	Objects look similar at any rotation (circles, symmetric patterns)

Warning

For aerial imagery where orientation carries semantic meaning (e.g., vehicle direction indicates movement), avoid rotation augmentation. A tank facing north is different from a tank facing south.

```yaml
# Aerial tank detection - orientation matters
degrees: 0.0

# General object detection
degrees: 10.0
```

9.2 translate

9.2.1 Definition

The `translate` parameter specifies the maximum random translation (shift) applied to images, as a fraction of image dimensions.

9.2.2 Purpose

Translation augmentation teaches the model to detect objects regardless of their position in the image frame. This prevents the model from learning position-dependent features.

9.2.3 Value Guidelines

Table 9.2: Translation Augmentation Values

translate	Shift Range	Use Case
0.0	None	Objects always centered
0.1	±10%	Standard for most detection tasks
0.2	±20%	Objects can appear anywhere in frame

```yaml
translate: 0.1  # Standard value
```

9.3 scale

9.3.1 Definition

The `scale` parameter specifies the range of random scaling (zoom) applied to images.

9.3.2 Purpose

Scale augmentation teaches the model to detect objects at various sizes, which is crucial when:

- Objects appear at different distances
- Camera zoom varies
- Object sizes naturally vary

9.3.3 Technical Background

When `scale: 0.5` is set, the scale factor is uniformly sampled from $[1-0.5, 1+0.5] = [0.5, 1.5]$. This means objects can appear at 50% to 150% of their original size.

9.3.4 Value Guidelines

Table 9.3: Scale Augmentation by Distance Variation

scale	Size Range	Use Case
0.0	100% only	Fixed distance, industrial inspection
0.2–0.3	70–130%	Controlled distance variation
0.5	50–150%	Standard drone surveillance (50–230m range)
0.7–0.9	10–190%	Extreme distance variation

```
# Drone surveillance with 50-230m range
scale: 0.5
```

9.4 fliplr (Horizontal Flip)

9.4.1 Definition

The `fliplr` parameter specifies the probability of flipping images horizontally (left-right mirror).

9.4.2 Purpose

Horizontal flipping doubles the effective variety of object poses in your dataset. Most objects look valid when mirrored horizontally.

9.4.3 Value Guidelines

Table 9.4: Horizontal Flip Settings

fliplr	Use Case
0.0	Text detection, asymmetric objects where left/right matters
0.5	Standard for most detection tasks

```
fliplr: 0.5  # Standard for vehicle/object detection
```

9.5 flipud (Vertical Flip)

9.5.1 Definition

The `flipud` parameter specifies the probability of flipping images vertically (up-down mirror).

9.5.2 Purpose

Vertical flipping is useful only when "up" and "down" are interchangeable in your images.

9.5.3 Value Guidelines

Table 9.5: Vertical Flip Settings

flipud	Use Case
0.0	Aerial imagery, street cameras, any scene with gravity
0.5	Microscopy, satellite, underwater scenes

Warning

For aerial/drone imagery, never use vertical flip. Tanks don't fly, and a vertically flipped image would show ground as sky.

```
# Aerial imagery - gravity matters
flipud: 0.0
```

Chapter 10

Color Augmentations

10.1 hsv_h (Hue)

10.1.1 Definition

The `hsv_h` parameter specifies the maximum random hue shift as a fraction of the full color wheel.

10.1.2 Purpose

Hue augmentation teaches the model to be robust to color variations:

- Different lighting conditions (daylight vs artificial)
- Camera color calibration differences
- Seasonal color changes (vegetation)

10.1.3 Technical Background

Hue represents the "color" component in HSV color space, measured as an angle on the color wheel (0–360). A value of `hsv_h: 0.015` allows shifts of ±1.5% of the color wheel, or approximately ±5.4.

10.1.4 Value Guidelines

Table 10.1: Hue Augmentation Settings

Image Type	hsv_h	Reasoning
Grayscale/Thermal	0.0	No color information exists
RGB - color matters	0.01–0.015	Subtle shifts preserve object identity
RGB - color invariant	0.05–0.1	Larger shifts when color is irrelevant

```
# RGB imagery
hsv_h: 0.015

# Grayscale/thermal imagery
hsv_h: 0.0
```

10.2 hsv_s (Saturation)

10.2.1 Definition

The `hsv_s` parameter specifies the maximum random saturation change as a fraction.

10.2.2 Purpose

Saturation augmentation teaches robustness to:

- Washed-out images (low saturation)
- Vivid images (high saturation)
- Different camera sensors

10.2.3 Value Guidelines

Table 10.2: Saturation Augmentation Settings

Image Type	hsv_s	Reasoning
Grayscale/Thermal	0.0	No saturation in grayscale
RGB - consistent lighting	0.4–0.5	Moderate variation
RGB - variable lighting	0.7	Standard for outdoor scenes

```
# RGB with variable outdoor lighting
hsv_s: 0.7

# Grayscale/thermal
hsv_s: 0.0
```

10.3 hsv_v (Value/Brightness)

10.3.1 Definition

The `hsv_v` parameter specifies the maximum random brightness change as a fraction.

10.3.2 Purpose

Brightness augmentation is critical for robustness to:

- Different times of day
- Shadow and sunlight variations
- Exposure differences
- Weather conditions

10.3.3 Value Guidelines

Table 10.3: Brightness Augmentation Settings

Lighting Conditions	hsv_v	Reasoning
Controlled/consistent	0.2–0.3	Small variation
Variable outdoor	0.4	Standard for most cases
Day/night variation	0.5–0.6	Large brightness range
Thermal/IR imagery	0.3–0.4	Temperature-based brightness varies

Note

Unlike hue and saturation, brightness augmentation applies to both RGB and grayscale images.

```
# Standard for most datasets
hsv_v: 0.4
```

Chapter 11

Advanced Augmentations

11.1 mosaic

11.1.1 Definition

The `mosaic` parameter specifies the probability of applying mosaic augmentation, which combines four training images into one.

11.1.2 Purpose

Mosaic augmentation is particularly powerful for:

- Small object detection (objects appear at various scales)
- Increasing batch variety
- Providing diverse context in each training image

11.1.3 Technical Background

Mosaic augmentation:

1. Selects 4 random training images
2. Scales and crops them
3. Combines them into a 2×2 grid
4. Adjusts bounding box coordinates accordingly

11.1.4 Value Guidelines

```
mosaic: 1.0  # Always apply during most of training
```

Tip

Always use `mosaic: 1.0` with `close_mosaic` set to disable it for the final epochs.

11.2 close_mosaic

11.2.1 Definition

The `close_mosaic` parameter specifies the number of epochs at the end of training to disable mosaic augmentation.

11.2.2 Purpose

Mosaic creates artificial image boundaries that don't exist in real images. Training on clean (non-mosaic) images at the end helps the model:

- Learn natural image boundaries
- Fine-tune on realistic data
- Improve final accuracy

11.2.3 Value Guidelines

$$\text{close\_mosaic} \approx \text{epochs} \times 0.01 \text{ to } 0.02 \tag{11.1}$$

Table 11.1: close_mosaic by Total Epochs

Total Epochs	close_mosaic
500	5–10
1000	10–15
1500	15–20
2000	20–30

```
# For 1000 epochs training
mosaic: 1.0
close_mosaic: 10
```

11.3 erasing

11.3.1 Definition

The `erasing` parameter specifies the probability of applying random erasing augmentation, which randomly occludes rectangular regions of the image.

11.3.2 Purpose

Random erasing improves model robustness by:

- Forcing the model to detect objects from partial views
- Simulating occlusions in real-world scenarios
- Preventing over-reliance on specific features

11.3.3 Value Guidelines

Table 11.2: Random Erasing Settings

erasing	Use Case
0.0	Objects are never occluded
0.2–0.3	Light occlusion robustness
0.4	Standard - good robustness
0.5–0.6	Strong - highly occluded environments

```
erasing: 0.4  # Standard for most detection tasks
```

Chapter 12

Augmentation Presets

This chapter provides complete augmentation configurations for common scenarios.

12.1 RGB Aerial/Drone Detection

```yaml
augmentation:
  # Geometric - conservative for aerial
  degrees: 0.0        # Orientation matters
  translate: 0.1      # Standard shift
  scale: 0.5          # Distance variation
  shear: 0.0          # No shear for top-down
  perspective: 0.0    # No perspective for nadir
  flipud: 0.0         # Gravity matters
  fliplr: 0.5         # Horizontal symmetry OK

  # Color - full RGB augmentation
  hsv_h: 0.015        # Subtle hue shift
  hsv_s: 0.7          # Variable saturation
  hsv_v: 0.4          # Brightness variation

  # Advanced
  mosaic: 1.0         # Always use mosaic
  mixup: 0.0          # Not needed
  copy_paste: 0.0     # Not needed
  erasing: 0.4        # Occlusion robustness
  close_mosaic: 10    # Disable at end
```

12.2 Grayscale/Thermal Detection

```yaml
augmentation:
  # Geometric - same as RGB
  degrees: 0.0
  translate: 0.1
  scale: 0.5
  shear: 0.0
  perspective: 0.0
  flipud: 0.0
  fliplr: 0.5

```

```
  # Color - no color augmentation
  hsv_h: 0.0          # No hue in grayscale
  hsv_s: 0.0          # No saturation in grayscale
  hsv_v: 0.4          # Brightness still applies

  # Advanced
  mosaic: 1.0
  mixup: 0.0
  copy_paste: 0.0
  erasing: 0.4
  close_mosaic: 10
```

12.3 Small Dataset (< 3000 images)

```
augmentation:
  # Geometric - more aggressive
  degrees: 10.0       # Allow some rotation
  translate: 0.15     # More shift
  scale: 0.6          # More scale variation
  shear: 5.0          # Light shear
  perspective: 0.0005  # Subtle perspective
  flipud: 0.0
  fliplr: 0.5

  # Color - more aggressive
  hsv_h: 0.02         # More hue variation
  hsv_s: 0.8          # More saturation
  hsv_v: 0.5          # More brightness

  # Advanced - use all techniques
  mosaic: 1.0
  mixup: 0.3          # Enable mixup for small data
  copy_paste: 0.2     # Enable copy-paste
  erasing: 0.5        # More erasing
  close_mosaic: 20
```

Chapter 13

Validation Configuration

13.1 patience (Early Stopping)

13.1.1 Definition

The `patience` parameter specifies the number of epochs to wait for improvement before automatically stopping training.

13.1.2 Purpose

Training should stop when:

- The model has converged (no more improvement possible)
- Overfitting begins (validation metrics worsen)
- Further training wastes computational resources

Early stopping monitors validation metrics and halts training if no improvement occurs for `patience` epochs.

13.1.3 Technical Background

The early stopping algorithm:

1. After each epoch, evaluate validation mAP
2. If mAP improves, save the model and reset patience counter
3. If mAP doesn't improve, increment patience counter
4. If patience counter reaches `patience`, stop training

13.1.4 Value Guidelines

$$\text{patience} \approx \text{epochs} \times 0.10 \text{ to } 0.15 \tag{13.1}$$

Table 13.1: Patience Values by Total Epochs

Total Epochs	patience
500	50–75
800	80–120
1000	100–150
1500	150–200
2000	200–300

Tip

Set epochs higher than you expect to need, and let patience stop training automatically. This ensures you don't stop too early while avoiding wasted computation.

```
# For 1000 epoch training
epochs: 1000
patience: 100  # Stop if no improvement for 100 epochs
```

Appendix A

Quick Reference Cheatsheet

A.1 Model Selection

Requirement	Model	Reason
Edge/mobile, fastest inference	YOLO12n	Smallest, fastest
Balanced speed/accuracy	YOLO12s	Good compromise
High accuracy, moderate speed	YOLO12m	Balanced for server deployment
High accuracy applications	YOLO12l	High quality detection
Maximum accuracy	YOLO12x	Best detection, cloud/server

A.2 Quick Configuration by Model

Model	Batch	LR	Warmup	Epochs	Patience
YOLO12n	160	0.02	5	1000	100
YOLO12s	64	0.01	5	1000	100
YOLO12m	32	0.008	8	1200	150
YOLO12l	16	0.004	10	1400	150
YOLO12x	8	0.002	12	1600	200

Values for RTX 5090 32GB VRAM, 5000 image dataset, raw training mode

A.3 RGB vs Grayscale Quick Reference

Parameter	RGB	Grayscale/Thermal
hsv_h	0.015	0.0
hsv_s	0.7	0.0
hsv_v	0.4	0.4
image_size	640×384 (16:9)	640×640 (1:1)

A.4 Dataset Size Quick Reference

Dataset Size	n	s	m	l	x
< 3,000 images	1500–2000	1500–2000	1800–2200	2000–2500	2200–2800
3,000–10,000 images	1000–1500	1000–1500	1200–1600	1400–1800	1600–2000
> 10,000 images	600–1000	600–1000	800–1200	1000–1400	1200–1600

Recommended epoch ranges by model size

Dataset Size	weight_decay
< 3,000 images	0.001
3,000–10,000 images	0.0005
> 10,000 images	0.0003

Recommended weight decay by dataset size

Appendix B

Troubleshooting Guide

B.1 Training Issues

Problem	Likely Cause	Solution
Loss = NaN	LR too high	Reduce LR by 50%, increase warmup
Loss oscillates	LR too high	Reduce LR by 30%
Loss stuck high	LR too low	Increase LR by 50%
Loss flat	Model converged or LR too low	Check if converged; if not, increase LR
Train ↓ Val ↑	Overfitting	More augmentation, higher weight_decay
OOM error	Batch too large	Reduce batch_size by 25%

B.2 Pre-Training Checklist

☐ `nc` matches in config, architecture, and dataset files

☐ Dataset folder exists with images and labels

☐ Learning rate scaled correctly for batch size

☐ Warmup epochs appropriate for LR and batch

☐ Augmentation settings match image type (RGB vs grayscale)

☐ Patience set to 10–15% of total epochs