YOLOv8 Implementation - Complete Documentation

📋 Table of Contents

🏗️ Architecture Overview

YOLOv8 is a single-stage object detector with three main parts:

```None
Input Image (640×640×3)
    ↓
┌──────────────────┐
│   BACKBONE       │  Extract features at multiple scales
│   (CSPDarknet)   │
└──────────────────┘
    ↓
┌──────────────────┐
│   NECK           │  Feature fusion (PAN-FPN)
│   (PAN-FPN)      │
└──────────────────┘
    ↓
┌──────────────────┐
│   HEAD           │  Predict boxes + classes
│   (Decoupled)    │
└──────────────────┘
    ↓
```

```
Predictions at 3 scales (P3, P4, P5)
```

🧱 Model Components

1. Basic Blocks (`yolov8_model.py`)

`Conv(c1, c2, k, s, p, g, act)`

Standard convolution block with BatchNorm and SiLU activation.

Parameters:

- `c1`: Input channels
- `c2`: Output channels
- `k`: Kernel size
- `s`: Stride
- `p`: Padding
- `g`: Groups (for grouped convolution)
- `act`: Whether to use activation

Flow:

```None
Input → Conv2d → BatchNorm2d → SiLU → Output
```

`Bottleneck(c1, c2, shortcut, g, e)`

Residual bottleneck block (like ResNet).

Parameters:

- `c1`: Input channels
- `c2`: Output channels
- `shortcut`: Use residual connection
- `g`: Groups
- `e`: Expansion ratio (default 0.5)

Flow:

```None
Input ──┬→ Conv(1×1) → Conv(3×3) ──┬→ Output
        │                          │
        └──────────────────────────┘  (if shortcut)
```

`C2f(c1, c2, n, shortcut, g, e)`

CSPNet-style block with split-concatenate architecture.

Parameters:

- `n`: Number of bottleneck layers
- Other params same as Bottleneck

Flow:

```None
Input → Conv(1×1) → Split into 2 parts
                       ↓              ↓
                    Part A    →    Bottleneck × n → Concat →
Conv(1×1) → Output
```

Why C2f?

- Reduces parameters while maintaining performance
- Better gradient flow through split connections

`SPPF(c1, c2, k)`

Spatial Pyramid Pooling - Fast version.

Flow:

```None
Input → Conv(1×1) → ┬→ MaxPool ─→ MaxPool ─→ MaxPool
                    │       ↓           ↓           ↓
                    └───────┴───────────┴───────────┴→ Concat →
Conv(1×1) → Output
```

Purpose: Captures multi-scale features efficiently.

---

2. Backbone (Feature Extraction)

The backbone extracts features at 5 scales (P1-P5):

Layer	Input Size	Output Size	Stride	Channels
P1	640×640×3	320×320	2	48
P2	320×320	160×160	4	96
P3	160×160	80×80	8	192
P4	80×80	40×40	16	384
P5	40×40	20×20	32	384

Code Flow:

```Python
x → Conv(3×3, s=2) → C2f → Conv(3×3, s=2) → C2f → ... → SPPF → P5
                      ↓                       ↓
                     P3                      P4
```

---

3. Neck (Feature Fusion)

PAN-FPN architecture combines features from different scales.

Top-Down Path (FPN):

```
None
P5 (20×20) —→ Upsample —→ Concat(P4) —→ C2f —→ P4_out
                                          ↓
                                   Upsample —→ Concat(P3) —→ C2f
—→ P3_out
```

Bottom-Up Path (PAN):

```
None
P3_out —→ Downsample —→ Concat(P4_out) —→ C2f —→ P4_final
                                            ↓
                                      Downsample —→ Concat(P5) —→
C2f —→ P5_final
```

Output Channels:

- P3: 256 channels (80×80)
- P4: 512 channels (40×40)
- P5: 1024 channels (20×20)

---

4. Detection Head

Decoupled head with separate branches for bbox regression and classification.

For each scale (P3, P4, P5):

```
None
Feature Map
    ↓
    ├─→ Conv → Conv → Conv → [4×reg_max] (bbox regression)
    │
    └─→ Conv → Conv → Conv → [nc] (classification)
```

Output per scale:

- **Bbox predictions:** `[B, 4×reg_max, H, W]` = 64 channels

- **Class predictions:** `[B, nc, H, W]` = 80 channels

Total anchors: 80×80 + 40×40 + 20×20 = 8,400 anchor points

🎯 Loss Function

Overview

YOLOv8 uses three loss components:

```None
Total Loss = λ₁×Box Loss + λ₂×Class Loss + λ₃×DFL Loss
           = 7.5×L_box    + 0.5×L_cls      + 1.5×L_dfl
```

---

1. Task-Aligned Assigner

Assigns ground truth boxes to anchor points based on alignment metric.

Algorithm:

```Python
# 1. Compute alignment metric for each GT-anchor pair
alignment = (cls_score^α) × (IoU^β)
            where α=0.5, β=6.0

# 2. Select top-k anchors per GT (k=10)
top_anchors = TopK(alignment, k=10)

# 3. Create foreground mask
fg_mask = anchors assigned to any GT

# 4. For each foreground anchor, match to best GT
matched_gt = argmax(IoU)
```

Why Task-Aligned?

- Considers both classification score AND localization quality
- More accurate than IoU-only matching
- Adaptive to model's current predictions

---

2. Box Loss (CIoU)

Complete IoU loss that considers:

- **Overlap:** Standard IoU
- **Distance:** Distance between box centers
- **Aspect ratio:** Consistency of w/h ratios

Formula:

```
None
CIoU = IoU - (ρ²/c²) - (α×v)

where:
  ρ² = distance between centers
  c² = diagonal of smallest enclosing box
  v = (4/π²) × (arctan(w_gt/h_gt) - arctan(w_pred/h_pred))²
  α = v / (1 - IoU + v)

Loss_box = mean(1 - CIoU)
```

Range:

- CIoU $\in [-1, 1]$
- Loss $\in [0, 2]$

---

3. Classification Loss (BCE)

Binary Cross-Entropy for each class independently.

Formula:

```
None
Loss_cls = BCE(pred_scores, target_scores) / score_sum

where:
  target_scores[i, j] = IoU_quality  if class matches
                      = 0            otherwise
```

Why IoU-weighted targets?

- Quality-aware: Better boxes get higher target scores
- Helps model learn confidence calibration

---

4. Distribution Focal Loss (DFL)

YOLOv8's key innovation for bbox regression.

Concept: Instead of directly regressing distances, predict a **distribution** over discrete bins.

How it works:

```
None
# Traditional: Predict single value
distance = model_output  # e.g., 5.7

# DFL: Predict probability distribution
probs = softmax(model_output)  # [p0, p1, ..., p15]
distance = Σ(i × probs[i])     # Expected value

# Loss: Cross-entropy on interpolated targets
target = 5.7 → interpolate between bins 5 and 6
  bin[5] weight = 0.3
  bin[6] weight = 0.7

Loss = 0.3×CE(probs, 5) + 0.7×CE(probs, 6)
```

Why DFL?

- More accurate than direct regression
- Better captures uncertainty
- Improves small object detection

Range: [0, reg_max-1] = [0, 15]

---

5. Box Encoding/Decoding

Encoding (GT boxes → DFL targets):

```python
# 1. Get anchor point (center of grid cell)
anchor = (grid_x + 0.5) × stride, (grid_y + 0.5) × stride

# 2. Compute distances in pixels
left = anchor_x - x1
top = anchor_y - y1
right = x2 - anchor_x
bottom = y2 - anchor_y

# 3. Normalize by stride to get DFL range [0, 15]
dist = [left, top, right, bottom] / stride
```

Decoding (DFL predictions → boxes):

```python
# 1. Get distribution from softmax
probs = softmax(pred_dist)  # [B, A, 4, 16]

# 2. Compute expected distance
dist = Σ(i × probs[i])  # [B, A, 4]

# 3. Scale by stride
dist_pixels = dist × stride

# 4. Convert to xyxy format
```

```
x1 = anchor_x - dist_pixels[0]
y1 = anchor_y - dist_pixels[1]
x2 = anchor_x + dist_pixels[2]
y2 = anchor_y + dist_pixels[3]
```

🔄 Training Pipeline

Data Flow

```None
1. Load Image + Labels
   ↓
2. Augmentation (Mosaic, HSV, Flip)
   ↓
3. Resize to 640×640
   ↓
4. Normalize to [0, 1]
   ↓
5. Batch Collation
   ├─ Images: [B, 3, 640, 640]
   └─ Targets: [N, 6] where N = total objects in batch
       Format: [batch_idx, class, x_center, y_center, width,
height]
              All coordinates normalized to [0, 1]
```

Training Loop

```Python
for epoch in range(epochs):
    # 1. Forward pass
    preds = model(images)  # List of [P3, P4, P5]

    # 2. Compute loss
```

```
    loss_dict = criterion(preds, targets)
    loss = loss_dict['total']

    # 3. Backward pass
    loss.backward()

    # 4. Gradient clipping (prevent explosion)
    clip_grad_norm_(model.parameters(), max_norm=10.0)

    # 5. Optimizer step
    optimizer.step()
    optimizer.zero_grad()

    # 6. EMA update (exponential moving average)
    ema.update(model)

    # 7. Learning rate scheduling
    scheduler.step()
```

Optimizations

1. Mixed Precision Training (AMP)

```python
with torch.cuda.amp.autocast():
    preds = model(imgs)
    loss = criterion(preds, targets)

scaler.scale(loss).backward()
scaler.step(optimizer)
scaler.update()
```

Benefits: 2x faster training, 50% less memory

2. Exponential Moving Average (EMA)

```python
ema_param = 0.9999 × ema_param + 0.0001 × model_param
```

Benefits: More stable and accurate final model

3. Warmup + Cosine Annealing

```
LR
 ↑
 |      ┌─────╮
 |     /       \
 |    /         \___
 |   /
 └──┴──────────────→ Epoch

   ↑       ↑
 Warmup  Cosine
```

📊 Data Processing

Mosaic Augmentation

Combines 4 images into one:

```
┌────────┬────────┐
│  Img1  │  Img2  │
│        │        │
├────────┼────────┤
│  Img3  │  Img4  │
│        │        │
└────────┴────────┘
```

Benefits:

- Learns from 4× more objects per batch

- Better small object detection
- Regularization effect

Label Format

Input format (YOLO .txt):

```None
class x_center y_center width height
0     0.5      0.5      0.3   0.4
```

All values normalized to [0, 1]

Internal format (after collation):

```Python
targets = torch.tensor([
    [batch_idx, class, x_center, y_center, width, height],
    [0,         0,     0.5,      0.5,      0.3,   0.4],
    [0,         1,     0.3,      0.7,      0.2,   0.3],
    [1,         2,     0.6,      0.4,      0.4,   0.4],
])
```

🔍 Inference Pipeline

```Python
# 1. Preprocess
img = cv2.resize(img, (640, 640))
img = torch.from_numpy(img).float() / 255.0

# 2. Forward pass
with torch.no_grad():
    preds = model(img)
```

```
# 3. Decode predictions
boxes, scores, classes = postprocess(preds)

# 4. Non-Maximum Suppression (NMS)
keep = nms(boxes, scores, iou_threshold=0.45)
final_boxes = boxes[keep]
final_scores = scores[keep]
final_classes = classes[keep]
```

📈 Model Sizes

Model	Depth	Width	Parameters	Speed
YOLOv8n	0.33	0.25	3.2M	Fastest
YOLOv8s	0.33	0.50	11.2M	Fast
YOLOv8m	0.67	0.75	25.9M	Balanced
YOLOv8l	1.00	1.00	43.7M	Accurate
YOLOv8x	1.00	1.25	68.2M	Most Accurate

🎓 Key Takeaways

1. **Architecture:** CSPDarknet backbone + PAN-FPN neck + Decoupled head
2. **Innovation:** Distribution Focal Loss for better bbox regression
3. **Assignment:** Task-aligned matching for optimal GT-anchor pairing
4. **Loss:** CIoU for localization + BCE for classification + DFL for distribution
5. **Training:** Heavy augmentation + EMA + Warmup + AMP for best results

---

📚 References

- YOLOv8 Paper: Ultralytics YOLOv8 Docs
- Distribution Focal Loss: Generalized Focal Loss
- Task Alignment: TOOD