Urban Scene Instance Segmentation

A Deep Dive into Computer Vision

Project Description & Technical Roadmap

Project Duration: 4 Weeks (12 hours/week = 48 hours total)
Domain: Urban Scenes (People + Vehicles)
Architecture: Mask R-CNN → Optimized for Real-time
Goal: Master instance segmentation from theory to deployment

February 1, 2026

Contents

1 Project Overview

1.1 Motivation

This project represents a strategic progression from object detection and tracking to instance segmentation, targeting urban scene understanding for applications in Physical AI, autonomous systems, and robotics. The focus on urban scenes provides:

- **Transferable Skills:** Techniques learned apply across all computer vision domains
- **Portfolio Value:** Direct relevance to autonomous vehicles, robotics, and smart city applications
- **Technical Depth:** Challenges including occlusion handling, small object detection, and real-time inference
- **Video Compatibility:** Natural extension to temporal tracking in video sequences

1.2 Background & Prerequisites

Technical Background

Prior Experience:

- Object detection and tracking (humans, vehicles)
- YOLOv8 architecture and PyTorch framework
- Video processing pipelines

Available Resources:

- Google Colab with limited GPU access
- 12 hours per week dedication
- 4-week project timeline

1.3 Project Goals

1. **Deep Understanding:** Master instance segmentation architecture from first principles
2. **Practical Implementation:** Build, train, and optimize a production-ready model
3. **Portfolio Development:** Create compelling demonstrations for technical resume
4. **Real-time Inference:** Achieve video-rate performance for practical applications

2 Instance Segmentation Fundamentals

2.1 The Problem Definition

Instance segmentation extends beyond traditional computer vision tasks by providing pixel-perfect masks for each individual object instance. The hierarchy of computer vision tasks:

Classification: "This image contains a person"

Object Detection: "There's a person at coordinates [x, y, w, h]"

Semantic Segmentation: "These pixels belong to 'person' class" (no instance distinction)

Instance Segmentation: "These specific pixels are person #1, those are person #2"

2.2 Key Challenges in Urban Scenes

1. **Occlusion:** People partially hidden by vehicles, overlapping pedestrians
2. **Scale Variation:** Distant cars (small) vs. nearby pedestrians (large)
3. **Class Diversity:** Multiple object classes (person, car, bicycle, truck, traffic light)
4. **Real-time Requirements:** Video applications demand fast inference (>15 FPS)
5. **Boundary Precision:** Accurate segmentation at object boundaries crucial for safety

3 Architectural Deep Dive: Mask R-CNN

3.1 Architecture Overview

Mask R-CNN employs a two-stage approach for instance segmentation:

3.2 Component Breakdown

3.2.1 1. Backbone Network (Feature Extraction)

Architecture: ResNet-50/101 + Feature Pyramid Network (FPN)
Purpose: Extract hierarchical visual features at multiple scales
Why FPN matters:

- Small objects require high-resolution features (fine details)
- Large objects require semantic features (contextual understanding)
- FPN provides both through multi-scale feature maps

Output: Feature maps at multiple scales: $\{P_2, P_3, P_4, P_5, P_6\}$

3.2.2 2. Region Proposal Network (RPN)

Purpose: Generate ~2000 candidate bounding boxes that might contain objects
Mechanism:

1. Slides a small network over feature maps
2. At each position, predicts 9 anchor boxes (3 scales × 3 aspect ratios)
3. Classifies each anchor: object vs. background

4. Refines anchor coordinates
5. Applies Non-Maximum Suppression (NMS) to reduce to ∼1000 proposals

Key Insight: RPN doesn't classify object types, only detects "objectness"

Common Misconception Clarified

Misconception: RPN proposes exactly one box per object
Reality: RPN over-proposes (∼1000 boxes), then NMS and classification filter to final detections
Example: For 3 people in an image, RPN might propose 100+ boxes, which get reduced to 3 final detections

3.2.3 3. RoI Align (Region of Interest Alignment)

Problem it solves: Previous RoI Pooling used pixel rounding, causing misalignment
Solution: Bilinear interpolation at exact (non-integer) coordinates
Mathematical difference:

$$\begin{aligned} \text{RoI Pooling:} \quad & [x = 10.7, y = 20.3] \rightarrow [x = 11, y = 20] \quad \text{(rounded)} \\ \text{RoI Align:} \quad & [x = 10.7, y = 20.3] \rightarrow \text{interpolated at exact location} \end{aligned}$$

Impact: Sharp, pixel-perfect mask boundaries (critical for overlapping objects)
Output: Fixed-size $7 \times 7 \times 256$ feature tensor per proposal

3.2.4 4. Detection Head (Three Parallel Branches)

For each RoI feature ($7 \times 7 \times 256$), three tasks execute simultaneously:

Branch A - Classification:
Input: $7 \times 7 \times 256$ features $\rightarrow$ Flatten $\rightarrow$ FC layers
Output: 81 class probabilities (80 COCO classes + background)
Example: [0.02, 0.05, 0.83, 0.01, ...] $\rightarrow$ Class 2 (bicycle)

Branch B - Bounding Box Regression:
Input: $7 \times 7 \times 256$ features $\rightarrow$ FC layers
Output: $[\Delta x, \Delta y, \Delta w, \Delta h]$ (box refinement offsets)
Purpose: Refines RPN proposal to better fit the object

Branch C - Mask Prediction:
Input: $7 \times 7 \times 256$ features $\rightarrow$ Conv layers $\rightarrow$ Upsample
Output: $[80, 28, 28]$ (80 class-specific masks at 28×28 resolution)
Critical: Only the mask for the predicted class is used!

3.3 The Mask Head: Class-Specific Masks

Critical Architectural Insight

The mask head generates **80 separate masks** (one per class), but only the mask corresponding to the predicted class is used during inference.
Why this design?

- Each mask specializes for its class (person mask learns person shapes, car mask learns car shapes)
- Handles overlapping objects elegantly (person and car in same RoI)
- The "person" mask ignores car pixels, even if visible in the RoI

Example: RoI contains 60% person + 40% bicycle

1. Classification predicts: "person" (highest confidence)
2. Mask head outputs 80 masks (including both person and bicycle masks)
3. We use only mask[person] → segments person pixels, ignores bicycle

3.4 Multi-Task Loss Function

Training optimizes three objectives simultaneously:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{cls}} + \mathcal{L}_{\text{box}} + \mathcal{L}_{\text{mask}} \tag{1}$$

1. **Classification Loss** ($\mathcal{L}_{\text{cls}}$): Cross-entropy

$$\mathcal{L}_{\text{cls}} = -\sum_{i=1}^{81} y_i \log(\hat{y}_i) \tag{2}$$

where y_i is ground truth and $\hat{y}_i$ is predicted probability for class i

2. **Bounding Box Loss** ($\mathcal{L}_{\text{box}}$): Smooth L1

$$\mathcal{L}_{\text{box}} = \sum_{i \in \{x,y,w,h\}} \text{smooth}_{L1}(t_i - t_i^*) \tag{3}$$

where t_i are predicted box parameters and t_i^* are ground truth

3. **Mask Loss** ($\mathcal{L}_{\text{mask}}$): Binary cross-entropy per pixel

$$\mathcal{L}_{\text{mask}} = -\frac{1}{m^2} \sum_{i=1}^{m} \sum_{j=1}^{m} [y_{ij} \log(\hat{y}_{ij}) + (1 - y_{ij}) \log(1 - \hat{y}_{ij})] \tag{4}$$

for a $m \times m$ mask (typically $m = 28$)

Critical detail: Mask loss only computed for the predicted class, not all 80 classes

4 Understanding Feature Maps: The Foundation

4.1 What is a Feature Map?

A **feature map** is a 3D tensor (array of numbers) that represents what the neural network "sees" at different layers of processing. It is NOT an abstract concept—it's actual data with a concrete structure.

Format:

$$\text{Feature Map Shape} = [\text{Height}, \text{Width}, \text{Channels}] \tag{5}$$

Example:

```
Feature Map: [50, 40, 256]
  - Height: 50 positions (rows)
  - Width: 40 positions (columns)
  - Channels: 256 feature detectors (depth)
```

Think of it as a **stack of images**, where each "layer" (channel) detects different visual patterns like edges, textures, or complex shapes.

4.2 From Input Image to Feature Maps

4.2.1 The Transformation Pipeline

Spatial Downsampling with Channel Expansion

As the image flows through convolutional layers:

- **Spatial dimensions** (height, width) decrease
- **Number of channels** increases
- Information becomes more **abstract** and **semantic**

Example progression:

Input Image:	$[640, 480, 3]$	
Conv Layer 1:	$[320, 240, 64]$	
Conv Layer 2:	$[160, 120, 128]$	
Conv Layer 3:	$[80, 60, 256]$	
Conv Layer 4:	$[40, 30, 512]$	$\leftarrow$ Used by RPN

What happened:

- Spatial resolution: $640 \times 480 \rightarrow 40 \times 30$ ($16\times$ reduction)
- Channels: $3 \rightarrow 512$ ($171\times$ increase)
- Each position now represents a larger region of the original image

4.3 Feature Map Positions and Receptive Fields

4.3.1 The Stride Concept

Each position in the feature map corresponds to a specific region in the original image. This mapping is determined by the **stride**:

$$\text{Stride} = \frac{\text{Input Image Dimension}}{\text{Feature Map Dimension}} \tag{6}$$

Example:

$$\text{Stride} = \frac{640}{40} = 16 \text{ pixels per position}$$

Position-to-Region Mapping:

$$\text{Position } (0,\ 0) \leftrightarrow \text{Image region } [0:16, 0:16]$$
$$\text{Position } (1,\ 0) \leftrightarrow \text{Image region } [16:32, 0:16]$$
$$\text{Position } (20,\ 15) \leftrightarrow \text{Image region } [320:336, 240:256]$$

Each position summarizes a 16×16 pixel region from the original image.

4.4 Understanding Channels: The Feature Detectors

4.4.1 What Does Each Channel Represent?

At each position in the feature map, there are 512 numbers (for a 512-channel feature map). Each number represents the activation of a different **learned feature detector**.

Example at position (20, 15):

```
Channel 0:   0.82  ← Strong vertical edge detected
Channel 1:   0.05  ← No horizontal edge
Channel 2:   0.91  ← Round shape (head-like) detected
Channel 3:   0.15  ← Weak diagonal pattern
Channel 4:   0.78  ← Person torso pattern detected
...
Channel 511: 0.44  ← Complex pattern partially present
```

4.4.2 Are Channels Predefined or Learned?

Critical Insight: Channels are LEARNED

Before Training: Channels contain random weights and detect nothing useful.
During Training: Through backpropagation on thousands of images, channels learn to specialize:

- Channel 12 might learn horizontal edges
- Channel 47 might learn vertical edges
- Channel 203 might learn complex "person-like" shapes

After Training: Channel weights are frozen and consistently detect their learned patterns.

These patterns are NOT programmed—they emerge from data!

4.4.3 Channel Flexibility: Not Locked to One Shape

A common misconception is that each channel detects exactly one rigid pattern. In reality, channels are **flexible**:

Channel 47 Example (Vertical Edge Detector):
Strong activation (0.9): Perfect vertical edge |
Medium activation (0.6): Slightly tilted edge /
Weak activation (0.2): Horizontal edge –
No activation (0.0): Uniform region

The channel responds with **varying degrees of activation** depending on how well the input matches its learned pattern. This flexibility allows the network to handle variations in:

- Lighting conditions
- Object orientations
- Partial occlusions
- Different materials and textures

4.4.4 Deep Layer Channels: Complex Pattern Detection

As you go deeper in the network, channels detect increasingly complex patterns:

Early layers: Simple patterns (edges, corners)

```
Channel 5: Vertical edges (3×3 filter)
```

Middle layers: Combinations of simple patterns

```
Channel 47: Parallel vertical edges || (5×5 filter)
```

Deep layers: High-level semantic patterns

```
Channel 203: Person-like shapes (head + vertical body)
   Activates for: standing person, person with raised arm
   Does not activate for: cars, trees
```

4.5 Multi-Scale Feature Maps: The Feature Pyramid Network

4.5.1 Why Multiple Scales?

Objects in images vary dramatically in size. A single-scale feature map cannot handle both small distant objects and large nearby objects effectively.

Feature Pyramid Network (FPN) Solution:

$$
\begin{aligned}
P_2: &\quad [160, 120, 256] \quad \text{(high resolution, small objects)} \\
P_3: &\quad [80, 60, 256] \quad \text{(medium resolution)} \\
P_4: &\quad [40, 30, 256] \quad \text{(lower resolution)} \\
P_5: &\quad [20, 15, 256] \quad \text{(lowest resolution, large objects)}
\end{aligned}
$$

Scale-Specific Detection:

- **Small objects** (distant car, 20×20 pixels): Use P_2 (high resolution)
- **Medium objects** (nearby person, 200×400 pixels): Use P_3, P_4
- **Large objects** (truck, 300×200 pixels): Use P_5 (low resolution)

RPN generates anchors at **all scales**, enabling detection of objects across a wide size range.

4.6 Concrete Numerical Example

Scenario: Feature map position (20, 15) in P_4 [40, 30, 512]

Step 1: Image Region Mapping

$$\begin{aligned} \text{Center}_x &= 20 \times 16 = 320 \text{ pixels} \\ \text{Center}_y &= 15 \times 16 = 240 \text{ pixels} \\ \text{Region} &= [320 : 336, 240 : 256] \quad (16 \times 16 \text{ patch}) \end{aligned}$$

This 16×16 pixel region in the original image is summarized by 512 feature values.

Step 2: Feature Vector at This Position

```
Features[20, 15, :] = [0.82, 0.15, 0.91, 0.05, ..., 0.44]
                        ↑     ↑     ↑     ↑          ↑
                       Ch0   Ch1   Ch2   Ch3   ... Ch511
```

Step 3: RPN Uses These Features

At this position, RPN:

1. Generates 9 anchor boxes (all centered at 320, 240)
2. For each anchor, uses the 512 features to predict:
   - Objectness score (is there an object?)
   - Box refinement (how to adjust the anchor?)

High activations in "person-related" channels → High objectness score → Proposal generated

5 Anchor Box Mechanics: Deep Dive

5.1 Anchor Generation at Each Position

5.1.1 The 9-Anchor Configuration

At **each position** in the feature map, RPN generates 9 anchor boxes by combining:

- **3 scales:** Small (128), Medium (256), Large (512) pixels
- **3 aspect ratios:** Square (1:1), Tall (1:2), Wide (2:1)
- $3 \times 3 = 9$ anchors per position

Critical Concept: Shared Center Point

All 9 anchors at a given position share the **same center point**, but differ in size and shape.

Example at position (20, 15):

- All 9 anchors centered at pixel (320, 240)
- Anchor 1: 128×128 box
- Anchor 5: 256×512 box
- Anchor 9: 1024×512 box

They are nested rectangles with a common center!

5.2 Anchor Box Coordinates: Actual Rectangles

Anchor boxes are **real geometric rectangles**, not abstract concepts. Each anchor is defined by 4 numbers:

$$\text{Anchor Box} = [x, y, \text{width}, \text{height}] \tag{7}$$

where:

- (x, y) = top-left corner coordinates (in pixels)
- width, height = box dimensions (in pixels)

5.2.1 Calculating Anchor Coordinates

Given:

- Position in feature map: $(p_x, p_y) = (20, 15)$
- Stride: 16 pixels
- Scale: 256 pixels
- Aspect ratio: 1:2 (tall)

Calculation:

$$\begin{aligned}
\text{Center}_x &= p_x \times \text{stride} = 20 \times 16 = 320 \\
\text{Center}_y &= p_y \times \text{stride} = 15 \times 16 = 240 \\
\text{Width} &= 256 \\
\text{Height} &= 256 \times 2 = 512 \quad \text{(aspect ratio 1:2)} \\
x &= \text{Center}_x - \frac{\text{Width}}{2} = 320 - 128 = 192 \\
y &= \text{Center}_y - \frac{\text{Height}}{2} = 240 - 256 = -16
\end{aligned}$$

Anchor Box: $[192, -16, 256, 512]$
Note: Negative y coordinate means the box extends above the image boundary—this is normal and handled during processing.

5.3 Anchors vs. Receptive Fields

Common Misconception

WRONG: Anchors must fit within the 16×16 receptive field.
CORRECT: Anchors are **centered** at the receptive field position but can be **much larger**.

The distinction:

- **Receptive field** (16×16): The image region the network examines to make predictions
- **Anchor box** (e.g., 512×512): The predicted size of the object that might be present

Analogy: You look through a small window (receptive field) and see part of an elephant's leg. You predict: "There's a large elephant here!" (large anchor box). The predicted box is much bigger than your viewing window.

5.4 Complete Anchor Grid Across Feature Map

For a feature map of size $[40, 30, 512]$:

$$\begin{aligned} \text{Positions} &= 40 \times 30 = 1{,}200 \\ \text{Anchors per position} &= 9 \\ \text{Total anchors} &= 1{,}200 \times 9 = 10{,}800 \end{aligned}$$

Distribution across image:

- Position (0, 0) → 9 anchors centered at (0, 0)
- Position (1, 0) → 9 anchors centered at (16, 0)
- Position (2, 0) → 9 anchors centered at (32, 0)
- ⋮
- Position (39, 29) → 9 anchors centered at (624, 464)

This dense grid ensures comprehensive coverage of potential object locations.

5.5 From Anchors to Proposals: The Refinement Process

5.5.1 Individual Anchor Processing

Each of the 10,800 anchors is evaluated independently:

1. **Objectness Classification:**

$$[\text{background\_score}, \text{object\_score}] = \text{RPN}(\text{features}) \tag{8}$$

2. **Box Refinement Prediction:**

$$[\Delta x, \Delta y, \Delta w, \Delta h] = \text{RPN}(\text{features}) \tag{9}$$

3. **Proposal Generation:**

$$\begin{aligned} x_{\text{proposal}} &= x_{\text{anchor}} + \Delta x \\ y_{\text{proposal}} &= y_{\text{anchor}} + \Delta y \\ w_{\text{proposal}} &= w_{\text{anchor}} + \Delta w \\ h_{\text{proposal}} &= h_{\text{anchor}} + \Delta h \end{aligned}$$

Key Insight: One anchor transforms into one proposal through independent refinement. Anchors are **NOT combined** to form proposals.

5.5.2 Filtering Pipeline

```
10,800 initial anchors
    ↓
Filter: Keep only object_score > 0.5
    ↓
~6,000 anchors remain
    ↓
Sort by objectness score (descending)
    ↓
Keep top 2,000 proposals
    ↓
Apply NMS (remove overlaps with IoU > 0.7)
    ↓
~1,000 final proposals → Detection Head
```

5.6 Handling Multiple Anchors on Same Object

Scenario: One person in image, multiple anchors detect it

```
Anchor 137: Covers person's upper body
  Objectness: 0.65 → Proposal 137

Anchor 142: Covers entire person (good fit!)
  Objectness: 0.95 → Proposal 142

Anchor 155: Covers person's lower body
  Objectness: 0.71 → Proposal 155
```

All three proposals overlap heavily on the same person.
NMS Resolution:

$$\text{IoU}(142, 137) = 0.78 > 0.7 \rightarrow \text{Keep 142 (higher score), remove 137}$$
$$\text{IoU}(142, 155) = 0.81 > 0.7 \rightarrow \text{Keep 142, remove 155}$$

Final result: Only Proposal 142 survives—the best-fitting anchor for the person.

6 Handling Overlapping Objects

6.1 The Complete Pipeline for Overlapping Instances

Scenario: Two people standing close together, partially overlapping

1. **Backbone + FPN:** Extract multi-scale features from entire image (once)
2. **RPN:** Proposes ~2000 boxes
   - Some boxes around person 1
   - Some boxes around person 2
   - Some boxes capturing both (these typically get filtered)
3. **NMS:** Reduces to ~1000 proposals (removes highly overlapping boxes with IoU > 0.7)
4. **RoI Align + Detection Head:** For each surviving proposal

- Extract $7 \times 7 \times 256$ features
- Classify, refine box, predict mask

5. **Post-processing:**
   - Keep only high-confidence detections (typically > 0.5)
   - Resize 28×28 masks to box dimensions
   - Paste masks into full image at box locations
   - Handle overlaps: higher confidence mask takes precedence OR alpha blending

6.2 Key Insights for Overlapping Objects

1. Independent Processing:
Each instance is segmented independently through its own RoI. The model doesn't explicitly reason about "person 1 vs person 2" boundaries—it predicts each mask in isolation.

2. Boundary Precision via RoI Align:
Sub-pixel alignment ensures overlapping boundaries are predicted at exact pixel locations, preventing blurry transitions.

3. Class-Specific Masks:
When a person overlaps with a car, the person mask ignores car pixels automatically because it's trained specifically for person shapes.

4. Failure Cases:
If RPN fails to propose separate boxes for overlapping objects, they may be detected as a single instance (rare with good RPN tuning).

7 Comparison: YOLO vs. Mask R-CNN

Aspect	YOLOv8 (Your Background)	Mask R-CNN
Architecture	Single-stage	Two-stage
Processing	One forward pass	Proposals → Per-RoI processing
Speed	Faster (>30 FPS)	Slower (∼5-10 FPS)
Accuracy	Good for detection	Better for segmentation
Overlapping objects	Struggles (grid-based)	Handles well (RoI-based)
Output	Bounding boxes	Boxes + pixel masks
Use case	Real-time detection	Accurate segmentation

Table 1: Architectural comparison between YOLO and Mask R-CNN

Why YOLO struggles with overlaps: Grid-based design means each grid cell can only predict a limited number of objects. When multiple objects occupy the same cell, conflicts arise.

Why Mask R-CNN excels: Each detected object gets dedicated processing through its own RoI, naturally handling overlaps.

8 4-Week Project Roadmap

8.1 Week 1: Foundation & Architecture Understanding (12 hours)

Goals:

- Deep understanding of Mask R-CNN architecture
- Environment setup and baseline model evaluation
- Visualization of internal components

Deliverables:

1. Google Colab environment with Detectron2 installed
2. Pretrained Mask R-CNN running on sample images
3. Visualizations of:
    - RPN proposals (∼1000 colored boxes on image)
    - Classification scores for each proposal
    - All 80 class-specific masks for selected proposals
    - Final segmentation output
4. Documented understanding of each component
5. Baseline metrics on Cityscapes validation set

Time Breakdown:

- Architecture study: 4 hours
- Environment setup: 2 hours
- Running baseline model: 2 hours
- Visualization implementation: 3 hours
- Documentation: 1 hour

8.2 Week 2: Custom Training Pipeline (12 hours)

Goals:

- Implement custom training loop
- Focus on person + vehicle classes (subset of COCO/Cityscapes)
- Understand hyperparameter effects

Deliverables:

1. Custom dataset loader for Cityscapes
2. Training pipeline with:
    - Data augmentation (rotation, scaling, color jitter)
    - Learning rate scheduling

- Checkpoint saving
- TensorBoard logging

3. Trained model on person + vehicle subset
4. Evaluation metrics:
   - mAP (mean Average Precision) for boxes
   - Mask IoU (Intersection over Union)
   - Per-class performance breakdown
5. Analysis of failure cases

Time Breakdown:

- Dataset preparation: 3 hours
- Training pipeline implementation: 4 hours
- Model training (on Colab GPU): 3 hours
- Evaluation and analysis: 2 hours

8.3 Week 3: Optimization & Video Extension (12 hours)

Goals:

- Optimize for real-time inference
- Extend to video sequences with temporal consistency
- Handle challenging cases (occlusions, small objects)

Deliverables:

1. Speed optimization:
   - Model quantization (FP32 $\rightarrow$ FP16/INT8)
   - Reduce RPN proposals (2000 $\rightarrow$ 500)
   - Batch processing for video frames
2. Temporal tracking:
   - Frame-to-frame instance matching
   - Temporal smoothing of masks
   - Handling appearance/disappearance
3. Video demo on urban traffic scenes
4. Performance metrics:
   - Inference speed (FPS)
   - Accuracy vs. speed trade-off analysis

Time Breakdown:

- Model optimization: 4 hours
- Temporal tracking implementation: 4 hours
- Video processing and testing: 3 hours
- Performance analysis: 1 hour

8.4 Week 4: Polish & Portfolio Preparation (12 hours)

Goals:

- Create compelling visualizations
- Comprehensive documentation
- Portfolio-ready presentation

Deliverables:

1. GitHub repository with:
   - Clean, documented code
   - README with project overview
   - Installation instructions
   - Usage examples
   - Model checkpoints (hosted)
2. Technical blog post / documentation:
   - Architecture explanation
   - Training process
   - Optimization techniques
   - Challenges and solutions
3. Demo materials:
   - Side-by-side comparison videos (input vs. output)
   - Interactive Colab notebook
   - Performance benchmarks
4. Presentation slides for portfolio

Time Breakdown:

- Code cleanup and documentation: 4 hours
- Visualization and demo creation: 4 hours
- Blog post / technical write-up: 3 hours
- Repository organization: 1 hour

9 Key Concepts & Common Misconceptions

9.1 Misconceptions Addressed

Misconception 1: Classification Branch and Pixels

Misconception: The classification branch analyzes individual pixels

Reality: Classification operates on the entire RoI feature vector ($7 \times 7 \times 256$ flattened), outputting one class per proposal. It doesn't see individual pixels—only high-level features.

Misconception 2: Mask Head Refines Boxes

Misconception: The mask head refines bounding boxes

Reality: Box refinement is handled by the box regression branch. The mask head only predicts segmentation masks on a fixed 28×28 grid.

Misconception 3: Single Mask Output

Misconception: Mask head outputs one mask per proposal

Reality: Mask head outputs 80 masks (one per class), but only the mask for the predicted class is used during inference. This allows class-specific mask learning.

Misconception 4: Exact Proposal Count

Misconception: RPN proposes exactly one box per object

Reality: RPN over-proposes ($\sim$1000-2000 boxes) for the entire image. NMS and classification filter these to final detections. Multiple proposals may overlap the same object initially.

9.2 Critical Concepts to Master

1. **Two-Stage Architecture:**
   - Stage 1: RPN proposes candidate regions
   - Stage 2: Each region processed independently
2. **Multi-Task Learning:**
   - Classification, box regression, and mask prediction trained jointly
   - Shared features improve all tasks
3. **Class-Specific Masks:**
   - 80 parallel mask predictions per RoI
   - Only the predicted class mask is used
   - Enables handling overlapping objects of different classes
4. **RoI Align Precision:**
   - Sub-pixel alignment via bilinear interpolation
   - Critical for sharp mask boundaries
   - Especially important for overlapping instances
5. **Feature Pyramid Networks:**
   - Multi-scale feature extraction
   - Small objects use high-res features
   - Large objects use semantic features

10 Expected Outcomes

10.1 Technical Skills Acquired

- Deep understanding of two-stage detection architectures
- Expertise in instance segmentation techniques
- PyTorch implementation skills for complex models
- Model optimization and deployment strategies
- Video processing and temporal tracking
- Debugging and failure case analysis

10.2 Portfolio Value

This project demonstrates:

1. **Technical Depth:** Beyond surface-level model usage to architectural understanding
2. **Practical Skills:** Training, optimization, and deployment pipeline
3. **Domain Expertise:** Urban scene understanding for Physical AI applications
4. **Problem-Solving:** Handling occlusions, overlaps, and real-time constraints
5. **Communication:** Clear documentation and visualization of complex concepts

10.3 Performance Targets

Metric	Baseline (Week 1)	Target (Week 4)
mAP (boxes)	∼35%	>40%
Mask IoU	∼32%	>38%
Inference Speed	∼3 FPS	>15 FPS
Video Temporal Consistency	N/A	Smooth tracking

Table 2: Performance targets across project timeline

11 Next Steps

11.1 Immediate Action Items

1. **Set up Google Colab environment**
   - Install PyTorch, Detectron2
   - Download Cityscapes dataset samples
   - Configure GPU runtime
2. **Load pretrained Mask R-CNN**
   - Use Detectron2's model zoo
   - Run inference on sample images

- Verify output format

3. **Implement visualization pipeline**
   - RPN proposal visualization
   - Classification score display
   - Mask overlay rendering

11.2 Long-term Extensions (Post-Project)

- Explore one-stage alternatives (YOLACT, SOLOv2)
- Implement panoptic segmentation (instance + semantic)
- Deploy on edge devices (NVIDIA Jetson)
- Contribute to open-source projects

12 References & Resources

12.1 Key Papers

1. He, K., et al. (2017). *Mask R-CNN*. ICCV 2017.
2. Lin, T.Y., et al. (2017). *Feature Pyramid Networks for Object Detection*. CVPR 2017.
3. Ren, S., et al. (2015). *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*. NeurIPS 2015.

12.2 Datasets

- **Cityscapes:** Urban street scenes (training set)
- **COCO:** Common objects in context (pre-training)
- **Mapillary Vistas:** Diverse street-level imagery (testing)

12.3 Tools & Frameworks

- **Detectron2:** Facebook's detection framework (primary)
- **PyTorch:** Deep learning framework
- **OpenCV:** Image and video processing
- **TensorBoard:** Training visualization

13 Conclusion

This project provides a structured path from theoretical understanding to practical mastery of instance segmentation. By focusing on urban scenes, you'll build skills directly applicable to cutting-edge applications in autonomous systems and Physical AI while creating a compelling portfolio piece that demonstrates both technical depth and practical implementation ability.

The four-week timeline balances comprehensive learning with realistic constraints, ensuring deep understanding without overwhelming complexity. Each week builds on previous knowledge,

culminating in a production-ready system that showcases your expertise to potential employers and collaborators.

"Understanding is not about memorizing architectures—it's about grasping the principles that make them work."