Part 2: Reasoning-Based Questions

Q1: Choosing the Right Approach

You are tasked with identifying whether a product is missing its label on an assembly line. The products are visually similar except for the label.

I would choose object detection as my primary approach since it can pinpoint exactly where labels should be on products and tell me if they're present or missing with specific confidence scores. This approach works great for assembly lines because I can process multiple products in a single camera frame and get precise bounding boxes that help document exactly which products have issues.

If detection struggles with tiny labels or poor lighting conditions, I'd fall back to a simpler binary classification approach where I first crop individual product regions, then classify each one as "has label" or "missing label." By combining both approaches, I could leverage detection's spatial awareness when it works well, but still maintain high accuracy through focused classification when needed. The key is that detection gives me more actionable information about where problems occur, which is valuable for quality control teams who need to quickly identify and fix labeling station issues.

Q2: Debugging a Poorly Performing Model

You trained a model on 1000 images, but it performs poorly on new images from the factory.

First, I'd visually compare my training images with the actual factory images side-by-side to spot obvious differences maybe the factory has different lighting, the camera angle changed, or products appear at different distances than my training data expected. I would then analyze my training data distribution to check if I'm missing certain product variants, time-of-day lighting conditions, or common factory scenarios like partial occlusions from workers' hands.

Next, I'd create a confusion matrix and manually review 50-100 failure cases to identify patterns - are errors happening with specific product types, certain times of day, or particular camera positions? I'd also run a quick experiment by taking just 100 factory images, fine-tuning my model on them, and seeing if performance jumps significantly, which would confirm I have a domain shift problem.

Finally, I'd use visualization tools like GradCAM to see what image features my model is actually looking at - it might be focusing on backgrounds or irrelevant details instead of the actual products.

Q3: Accuracy vs Real Risk

Your model has 98% accuracy but still misses 1 out of 10 defective products.

Accuracy is absolutely the wrong metric here because missing defective products could mean shipping dangerous items to customers, leading to recalls, lawsuits, or serious safety incidents way worse than occasionally flagging a good product as defective.

What I really need to focus on is recall for the defective class, which tells me exactly what percentage of defective products I'm catching, and right now at 90% (9 out of 10), that's probably unacceptable for most industries.

I would implement a cost-based evaluation where missing a defect might cost $10,000 in recalls while a false alarm only costs $10 in re-inspection time, then optimize my model based on minimizing total cost rather than maximizing accuracy. I'd also adjust my model's decision threshold to be much more sensitive to potential defects - I'd rather have operators manually check 100 good products than let one defective product slip through to customers. For critical safety applications, I might even implement a dual-model system where any product flagged as potentially defective by either model gets inspected.

Q4: Annotation Edge Cases

You're labeling data, but many images contain blurry or partially visible objects.

I'd definitely keep these messy images because real factory floors are chaotic conveyor belts cause motion blur, workers' hands block the camera, and products appear at weird angles, so training only on perfect photos would make my model useless in production.

My approach would be to label these tricky cases with extra tags like "blurry" or "partially_visible" so I know exactly what challenges each image presents and can adjust training accordingly. Sure, including this messy data might drop my validation accuracy from 95% to maybe 90%, but I'd rather have a model that honestly performs at 90% everywhere than one that claims 95% in the lab but crashes to 70% on the factory floor. The smart move is to train in stages - start with clear images so the model learns what to look for, then gradually add the challenging stuff to make it robust. I'd also create two test sets (one clean, one messy) so I can tell management "expect 95% accuracy on good days, 85% when things get hectic" - being transparent about real-world performance beats overpromising every time.