Computer Vision Engineer

Interview Questions and Answers

Core Concepts

This section focuses on fundamental principles and advanced concepts that an experienced developer should master.

1. Explain the difference between SIFT and SURF feature detection algorithms and their practical applications.

SIFT (Scale-Invariant Feature Transform) and **SURF (Speeded-Up Robust Features)** are both local feature detectors and descriptors, but differ in key aspects:

SIFT:

- More accurate and robust to scale/rotation changes
- Computationally more expensive
- Uses Difference of Gaussian (DoG) for feature detection
- 128-dimensional feature descriptor

SURF:

- Faster computation using box filters and integral images
- 64-dimensional feature descriptor
- Slightly less accurate but suitable for real-time applications

2. How would you implement a real-time object tracking system using Kalman filtering?

Implementation Approach:

```
def kalman_tracker(measurement):
    # State transition matrix
    F = np.array([[1, 0, dt, 0], [0, 1, 0, dt], [0, 0, 1, 0], [0, 0, 0, 1]])
    # Measurement matrix
    H = np.array([[1, 0, 0, 0], [0, 1, 0, 0]])
    # Predict
    x = F @ x_prev
    P = F @ P_prev @ F.T + Q
    # Update
    K = P @ H.T @ np.linalg.inv(H @ P @ H.T + R)
```

- Initialize state vector with position and velocity
- Predict object's next position using motion model
- Update predictions with new measurements
- Handle occlusions using prediction step

3. Describe the architecture and working principles of Mask R-CNN for instance segmentation.

Mask R-CNN Architecture Components:

- **Backbone Network:** Usually ResNet + FPN for feature extraction
- **Region Proposal Network (RPN):** Generates region proposals
- **RoI Align:** Precisely aligns extracted features with input
- **Mask Branch:** FCN for pixel-wise segmentation

Working Process:

- Extract features using backbone network
- Generate region proposals using RPN
- Perform classification and bounding box regression
- Generate binary mask for each RoI

4. How would you handle class imbalance in a deep learning-based object detection system?

Strategies for Handling Class Imbalance:

- **Data-level solutions:**
  - Oversampling minority classes
  - Undersampling majority classes
  - Data augmentation techniques
- **Algorithm-level solutions:**
  - Focal Loss implementation
  - Class-weighted loss functions
  - Hard negative mining

```
def focal_loss(y_true, y_pred, gamma=2.0):
    ce_loss = tf.keras.losses.binary_crossentropy(y_true, y_pred)
    pt = tf.exp(-ce_loss)
    focal_loss = (1 - pt) ** gamma * ce_loss
    return focal_loss
```

5. Explain the concept of Feature Pyramid Networks (FPN) and their advantages in object detection.

FPN Architecture:

- **Top-down pathway:** High-level semantic features
- **Bottom-up pathway:** High-resolution features
- **Lateral connections:** Merge features from both pathways

Advantages:

- Multi-scale feature representation
- Better detection of objects at different scales
- Improved small object detection
- Maintains semantic information across scales

Implementation:

```
def build_fpn(C2, C3, C4, C5):
    P5 = Conv2D(256, (1,1))(C5)
    P4 = Add()([UpSampling2D()(P5), Conv2D(256, (1,1))(C4)])
    P3 = Add()([UpSampling2D()(P4), Conv2D(256, (1,1))(C3)])
```

6. How would you implement an efficient image similarity search system for large-scale datasets?

Implementation Strategy:

- **Feature Extraction:**
  - Use pre-trained CNN for embedding generation
  - Dimension reduction using PCA if needed
- **Indexing:**
  - Approximate Nearest Neighbors (ANN)
  - Locality-Sensitive Hashing (LSH)

```
def build_index(images):
    model = load_pretrained_model()
    features = model.predict(images)
    index = faiss.IndexFlatL2(features.shape[1])
    index.add(features)
    return index
```

7. Describe the challenges and solutions in implementing real-time semantic segmentation.

Key Challenges:

- **Computational Efficiency:**
  - Model architecture optimization
  - Hardware acceleration (GPU/TPU)
- **Accuracy vs Speed Trade-off:**
  - Lightweight architectures (ENet, FastSCNN)
  - Model pruning and quantization

Solutions:

```
def build_lightweight_segmentation():
    return Sequential([
        MobileNetV2(include_top=False),
        ASPP(dilation_rates=[6, 12, 18]),
        UpSampling2D(size=(4, 4))
    ])
```

8. How would you implement and optimize a deep learning-based face recognition system?

Implementation Steps:

- **Face Detection:** Use MTCNN or RetinaFace
- **Face Alignment:** Landmark detection and alignment
- **Feature Extraction:** Deep CNN (e.g., FaceNet, ArcFace)
- **Similarity Comparison:** Cosine similarity or L2 distance

```
def face_recognition_pipeline(image):
    faces = detector.detect_faces(image)
    embeddings = []
    for face in faces:
        aligned = align_face(face)
        embedding = model.get_embedding(aligned)
        embeddings.append(embedding)
```

9. Explain the concept and implementation of attention mechanisms in vision transformers.

Vision Transformer Components:

- **Patch Embedding:** Split image into fixed-size patches
- **Positional Encoding:** Add position information
- **Self-Attention:** Compute attention between patches

```
def self_attention(q, k, v):
    scores = torch.matmul(q, k.transpose(-2, -1)) / math.sqrt(d_k)
    attention = F.softmax(scores, dim=-1)
    return torch.matmul(attention, v)
```

Key Advantages:

- Global context modeling
- Position-aware processing
- Parallel computation

10. How would you design a system for 3D object reconstruction from multiple 2D images?

System Components:

- **Feature Extraction:**
  - SIFT/SURF for keypoint detection
  - Feature matching across views
- **Structure from Motion (SfM):**
  - Camera pose estimation
  - 3D point cloud generation

```
def reconstruct_3d(images):
    points_2d = extract_features(images)
```

```
camera_matrices = estimate_camera_poses(points_2d)
point_cloud = triangulate_points(points_2d, camera_matrices)
mesh = poisson_surface_reconstruction(point_cloud)
```

Data Structures and Algorithms

Questions in this section test your understanding of how to work with and manipulate data efficiently.

1. How would you implement an LRU (Least Recently Used) Cache with O(1) time complexity for both get and put operations?

Key Implementation Points:

- Use HashMap for O(1) lookups
- Use Doubly Linked List for O(1) removals/insertions
- Track capacity limit

```
class LRUCache:
    def __init__(self, capacity):
        self.cache = {}
        self.capacity = capacity
        self.dll = DoublyLinkedList()

    def get(self, key):
        if key in self.cache:
            self.dll.move_to_front(self.cache[key])
            return self.cache[key].value
```

2. Explain how you would implement a sliding window maximum algorithm efficiently

Optimal Approach:

- Use a deque to maintain indices of potential maxima
- Time Complexity: O(n)
- Space Complexity: O(k) where k is window size

```
def maxSlidingWindow(nums, k):
    result = []
    deq = collections.deque()
    for i in range(len(nums)):
        while deq and deq[0] < i - k + 1:
            deq.popleft()
        while deq and nums[deq[-1]] < nums[i]:
            deq.pop()
        deq.append(i)
```

3. How would you design an efficient algorithm to find all pairs of integers in an array that sum to a given target?

Two-Pointer Approach:

- Sort array first: O(nlogn)
- Use two pointers from start and end
- Time Complexity: O(nlogn)
- Space Complexity: O(1)

```
def findPairs(nums, target):
    nums.sort()
    left, right = 0, len(nums) - 1
    while left < right:
        curr_sum = nums[left] + nums[right]
        if curr_sum == target:
            yield (nums[left], nums[right])
```

4. Explain how you would implement a thread-safe circular buffer/ring buffer

Key Components:

- Fixed-size array
- Head and tail pointers
- Thread synchronization mechanism

```
class CircularBuffer:
    def __init__(self, size):
        self.buffer = [None] * size
        self.head = self.tail = 0
        self.lock = threading.Lock()
        self.not_full = threading.Condition(self.lock)
        self.not_empty = threading.Condition(self.lock)
```

5. How would you implement a concurrent hash map with fine-grained locking?

Implementation Strategy:

- Segment the hash table into multiple parts
- Use separate locks for each segment
- Implement lock striping

```
class ConcurrentHashMap:
    def __init__(self, segments=16):
        self.segments = [[] for _ in range(segments)]
        self.locks = [threading.Lock() for _ in range(segments)]

    def get_segment(self, key):
        return hash(key) % len(self.segments)
```

6. Explain how you would implement a trie (prefix tree) for efficient string searches

Key Features:

- Node structure with character and children map
- Methods for insert and search
- Prefix matching capability

```
class TrieNode:
    def __init__(self):
        self.children = {}
        self.is_end = False

    def insert(self, word):
        node = self
        for char in word:
            node.children.setdefault(char, TrieNode())
            node = node.children[char]
```

7. How would you implement a min-heap with decrease-key operation?

Implementation Details:

- Array-based binary heap
- Index mapping for O(1) element location
- Bubble-up operation for decrease-key

```
class MinHeap:
    def __init__(self):
        self.heap = []
        self.pos = {}

    def decrease_key(self, item, new_key):
        pos = self.pos[item]
        self.heap[pos] = new_key
        self._bubble_up(pos)
```

8. Explain how to implement a skip list for efficient searching

Key Concepts:

- Probabilistic data structure
- Multiple layers of linked lists
- O(log n) average search time

```
class SkipNode:
    def __init__(self, level, key, value):
        self.key = key
        self.value = value
        self.forward = [None] * (level + 1)

    def search(self, key):
        current = self.head
```

9. How would you implement an efficient algorithm for finding the k most frequent elements in a stream?

Solution Approach:

- Use HashMap for frequency counting
- Maintain min-heap of size k
- Time Complexity: O(n log k)

```
def topKFrequent(nums, k):
    count = collections.Counter(nums)
    heap = []
    for num, freq in count.items():
        heapq.heappush(heap, (freq, num))
        if len(heap) > k:
            heapq.heappop(heap)
```

10. Explain how to implement a thread-safe producer-consumer queue with blocking operations

Implementation Requirements:

- Thread-safe queue implementation
- Blocking put/get operations
- Size limit handling

```
class BlockingQueue:
    def __init__(self, maxsize):
        self.queue = collections.deque()
        self.maxsize = maxsize
        self.lock = threading.Lock()
        self.not_full = threading.Condition(self.lock)
        self.not_empty = threading.Condition(self.lock)
```

System Design

These questions evaluate your ability to think about the bigger picture, including architecture, scalability, and performance.

1. Design a scalable computer vision system for real-time object detection in surveillance cameras

Key Components and Considerations:

- **Input Processing:** Distributed video ingestion system to handle multiple camera streams
- **Processing Pipeline:** Frame extraction, preprocessing, object detection model inference
- **Architecture:**
  - Edge computing for initial processing
  - Cloud backend for complex analytics
  - Message queuing system (Kafka/RabbitMQ) for stream processing
- **Scalability:**
  - Horizontal scaling of inference servers
  - Load balancing using nginx/HAProxy
  - GPU cluster management
- **Storage:**
  - Object storage for video segments
  - Time-series DB for metadata
  - Cache layer for frequent access patterns

2. How would you design a face recognition system for a large enterprise with 100,000 employees?

System Architecture:

- **Data Management:**
  - Face embedding database using vector similarity search (FAISS)
  - Distributed cache for frequent matches
- **Processing Pipeline:**
  - Face detection
  - Face alignment and normalization
  - Feature extraction using deep learning
  - Similarity matching
- **Performance Optimization:**
  - Batch processing for registration
  - Index sharding for parallel search
  - Result caching
- **Scalability:**
  - Microservices architecture
  - Load-balanced inference servers
  - Distributed database system

3. Design a content moderation system that can process millions of images daily

System Components:

- **Input Processing:**
  - S3/blob storage for images
  - Queue-based processing system
- **Classification Pipeline:**
  - Multiple ML models for different violation types
  - Ensemble decision making
  - Human review integration
- **Architecture:**
  - Serverless functions for preprocessing
  - Container-based ML inference

    - Results aggregation service
- **Monitoring:**
    - Model performance metrics
    - Processing latency tracking
    - Error rate monitoring

4. How would you design a real-time video analytics system for retail stores?

Design Components:

- **Edge Processing:**
    - On-premise video processing units
    - Local object detection and tracking
- **Analytics Pipeline:**
    - Customer counting
    - Heat map generation
    - Dwell time analysis
- **Data Architecture:**
    - Time-series database for metrics
    - Graph database for trajectory analysis
    - Real-time dashboard updates
- **Scaling Strategy:**
    - Edge-cloud hybrid architecture
    - Batch analytics for historical data
    - Real-time streaming for alerts

5. Design a system for automatic license plate recognition (ALPR) for a city-wide traffic management system

System Design:

- **Camera Integration:**
    - Distributed camera network
    - Edge processing units
- **Processing Pipeline:**
    - Plate detection
    - Character segmentation
    - OCR processing
- **Data Management:**
    - Distributed database for plate records
    - In-memory cache for hot data
    - Data retention policies
- **Integration:**
    - REST APIs for external systems
    - Real-time notification system
    - Batch processing for analytics

6. How would you design a visual search system similar to Google Lens?

Architecture Components:

- **Image Processing:**
    - Feature extraction using CNN
    - Multi-modal embedding generation
- **Search Infrastructure:**
    - Vector similarity search (Elasticsearch/FAISS)
    - Category-specific indices
- **Backend Services:**
    - Image preprocessing service
    - Recognition services (text, objects, landmarks)
    - Search orchestration service
- **Scalability:**
    - Distributed feature extraction
    - Sharded search indices
    - Caching layer for popular queries

7. Design a real-time facial expression analysis system for video conferencing

System Components:

- **Client-Side:**
  - WebRTC integration
  - Frame extraction
  - Local preprocessing
- **Server Architecture:**
  - WebSocket servers for real-time communication
  - GPU-enabled inference servers
  - Expression classification service
- **Data Flow:**
  - Frame sampling strategy
  - Batch processing for efficiency
  - Result aggregation and smoothing
- **Performance:**
  - Load balancing
  - Auto-scaling rules
  - Fallback mechanisms

8. Design a document scanning and OCR system that can handle millions of documents daily

System Architecture:

- **Input Processing:**
  - Multi-format document support
  - Image enhancement pipeline
- **OCR Pipeline:**
  - Layout analysis
  - Text detection and recognition
  - Post-processing and validation
- **Infrastructure:**
  - Queue-based processing
  - Distributed OCR workers
  - Result aggregation service
- **Storage:**
  - Document store (MongoDB)
  - Full-text search (Elasticsearch)
  - Blob storage for originals

9. How would you design a real-time quality control system for a manufacturing line using computer vision?

Design Elements:

- **Image Acquisition:**
  - High-speed industrial cameras
  - Lighting control system
- **Processing Pipeline:**
  - Real-time defect detection
  - Measurement and verification
  - Quality classification
- **System Architecture:**
  - Edge processing units
  - Real-time decision making
  - Central monitoring system
- **Integration:**
  - PLC communication
  - Manufacturing execution system (MES)
  - Quality management system (QMS)

10. Design a large-scale image classification system that can handle various AI models and be easily updated

System Design:

- **Model Management:**

    - Model versioning system
    - A/B testing framework
    - Model registry service
- **Inference Architecture:**
    - Model serving platform (TensorFlow Serving/Triton)
    - Load balancing and auto-scaling
    - Feature store integration
- **Pipeline Components:**
    - Input validation and preprocessing
    - Multi-model inference orchestration
    - Result aggregation and post-processing
- **Monitoring:**
    - Model performance metrics
    - System health monitoring
    - Alert management

Coding and Debugging

This section presents practical coding challenges and questions about debugging techniques.

1. How would you implement histogram equalization for image enhancement in OpenCV?

Key Components:

- Calculate histogram of input image
- Normalize cumulative distribution
- Apply transformation

```
def hist_equalize(img):
    hist = cv2.calcHist([img], [0], None, [256], [0,256])
    cdf = hist.cumsum()
    cdf_normalized = cdf * hist.max() / cdf.max()
    return cv2.LUT(img, cdf_normalized)
```

Note: OpenCV provides cv2.equalizeHist() for direct implementation.

2. Explain how you would detect and match features between two images using SIFT?

Implementation Steps:

```
sift = cv2.SIFT_create()
kp1, des1 = sift.detectAndCompute(img1, None)
kp2, des2 = sift.detectAndCompute(img2, None)
bf = cv2.BFMatcher()
matches = bf.knnMatch(des1, des2, k=2)
good = [m for m,n in matches if m.distance < 0.75*n.distance]
```

Key points:

- SIFT is scale and rotation invariant
- Use ratio test for better matches
- Consider FLANN for larger datasets

3. How would you implement a real-time object detector using a sliding window approach?

Implementation:

```
def sliding_window(image, step, window):
    for y in range(0, image.shape[0] - window[1], step):
        for x in range(0, image.shape[1] - window[0], step):
            yield (x, y, image[y:y+window[1], x:x+window[0]])
```

Optimization tips:

- Use image pyramids for multi-scale detection
- Implement non-maximum suppression
- Consider using integral images for faster computation

4. Explain how you would implement image segmentation using watershed algorithm?

Implementation Steps:

```
markers = np.zeros(img.shape[:2], dtype=np.int32)
markers[sure_fg] = 2
markers[sure_bg] = 1
markers = cv2.watershed(img, markers)
```

```
img[markers == -1] = [255,0,0]
```

Key considerations:

- Proper marker selection is crucial
- Pre-process with noise removal
- Post-process to handle over-segmentation
- Consider marker-controlled watershed for better results

5. How would you implement camera calibration in OpenCV?

Implementation:

```
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 30, 0.001)
ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(
    objpoints, imgpoints, gray.shape[::-1], None, None)
```

Required steps:

- Collect calibration pattern images
- Find chessboard corners
- Prepare object points
- Calculate camera matrix and distortion coefficients

6. Explain how you would implement optical flow using Lucas-Kanade method?

Implementation:

```
lk_params = dict(winSize=(15,15), maxLevel=2)
new_points, status, error = cv2.calcOpticalFlowPyrLK(
    old_frame, frame, points, None, **lk_params)
```

Important considerations:

- Select good features to track
- Handle point validation
- Consider pyramid levels for handling large motions
- Implement proper error handling for lost points

7. How would you implement image stitching for panorama creation?

Key Steps:

```
stitcher = cv2.Stitcher.create()
status, panorama = stitcher.stitch(images)
if status != cv2.Stitcher_OK:
    print('Stitching failed!')
```

Implementation details:

- Feature detection and matching
- RANSAC for homography estimation
- Bundle adjustment
- Image blending techniques
- Handle exposure differences

8. Explain how you would implement facial landmark detection using dlib?

Implementation:

```
detector = dlib.get_frontal_face_detector()
predictor = dlib.shape_predictor('shape_predictor_68_face_landmarks.dat')
landmarks = predictor(image, face)
```

Key aspects:

- Face detection preprocessing
- Landmark model selection
- Point extraction and normalization

- Error handling for multiple faces

9. How would you implement image super-resolution using deep learning?

Implementation approach:

```
model = Sequential([
    Conv2D(64, (3,3), padding='same', activation='relu'),
    Conv2D(32, (3,3), padding='same', activation='relu'),
    Conv2D(3, (3,3), padding='same')])
```

Key considerations:

- Dataset preparation and augmentation
- Loss function selection (MSE vs perceptual)
- Architecture choice (SRCNN, ESRGAN)
- Training strategy and validation

10. Explain how you would implement real-time pose estimation?

Implementation:

```
net = cv2.dnn.readNetFromTensorflow('graph_opt.pb')
net.setInput(cv2.dnn.blobFromImage(frame, 1.0, (368,368)))
out = net.forward()
```

Important aspects:

- Model selection (OpenPose, PoseNet)
- Real-time optimization techniques
- Keypoint extraction and filtering
- Handling multiple people in frame
- Post-processing for smooth tracking

Behavioral Questions

These questions assess your soft skills, problem-solving approach, and how you work in a team.

1. Tell me about a challenging computer vision project you worked on and how you overcame technical obstacles.

Situation: At my previous role, we needed to develop a real-time facial recognition system for a high-security facility with 99.9% accuracy requirements.

Task: I was responsible for improving the system's performance, which was only achieving 95% accuracy and suffering from false positives.

Action: I implemented a multi-stage verification pipeline using a combination of deep learning models (MTCNN for detection, FaceNet for embeddings) and added additional preprocessing steps for varying lighting conditions. I also introduced an ensemble approach combining multiple models' predictions.

Result: The system achieved 99.95% accuracy in production, reduced false positives by 98%, and processed frames 30% faster than the previous implementation.

2. Describe a situation where you had to optimize a computer vision algorithm for better performance.

Situation: Our object detection system was taking too long to process surveillance camera feeds, causing significant delays.

Task: I needed to optimize the pipeline to process 30 FPS without compromising accuracy.

Action: I profiled the code and identified bottlenecks. Implemented GPU acceleration for preprocessing, used TensorRT for model optimization, and introduced batch processing. Also implemented frame-skipping during non-critical periods.

Result: Achieved real-time processing at 30 FPS with only a 0.5% drop in accuracy. The optimizations reduced CPU usage by 60% and GPU memory consumption by 40%.

3. Tell me about a time when you had to make a difficult technical decision that impacted the team.

Situation: Our team was divided between using traditional computer vision methods versus deep learning for a critical medical imaging project.

Task: As the technical lead, I needed to evaluate both approaches and make a decision that balanced accuracy, development time, and maintainability.

Action: I organized a proof-of-concept sprint where we tested both approaches with real data. I documented the pros and cons, considering factors like inference speed, explainability, and training data requirements.

Result: We chose a hybrid approach that used classical CV for preprocessing and deep learning for final classification. This decision led to 95% accuracy while maintaining interpretability for medical professionals.

4. How do you handle disagreements with team members about technical approaches?

Situation: A senior colleague strongly advocated for using a complex YOLO variant for a simple object counting task.

Task: I needed to convince the team that a simpler approach would be more appropriate without creating tension.

Action: I prepared a detailed comparison document with benchmarks, maintenance costs, and

deployment considerations. I organized a team discussion where everyone could share their concerns and suggestions.

Result: The team agreed to start with a simpler solution (basic CNN + traditional CV) and evolve if needed. This approach met all requirements while being easier to maintain and deploy.

5. Describe a situation where you had to mentor a junior developer in computer vision concepts.

Situation: A junior developer was struggling with implementing image segmentation algorithms.

Task: I needed to help them understand both theoretical concepts and practical implementation while maintaining their confidence.

Action: I created a structured learning plan, starting with basic concepts like edge detection and gradually moving to advanced topics. We had regular pair programming sessions and code reviews.

Result: Within three months, they successfully implemented a U-Net architecture for semantic segmentation and contributed valuable improvements to our image processing pipeline.

6. Tell me about a time when you had to deal with incomplete or noisy training data.

Situation: We received a dataset for facial emotion recognition that was poorly labeled and had inconsistent image quality.

Task: I needed to develop a robust model despite data quality issues within a tight deadline.

Action: I implemented automated data cleaning pipelines, used data augmentation techniques, and created a semi-supervised learning approach to leverage unlabeled data. Also developed quality metrics to filter out unreliable samples.

Result: The final model achieved 88% accuracy on the test set, comparable to models trained on clean datasets. The data processing pipeline became a standard tool for future projects.

7. How do you handle project deadlines when dealing with experimental computer vision techniques?

Situation: We were implementing a novel pose estimation algorithm with uncertain development time.

Task: I needed to manage stakeholder expectations while exploring cutting-edge techniques.

Action: I broke down the project into measurable milestones, maintained a parallel development track with a proven fallback solution, and provided weekly progress updates with clear metrics.

Result: We delivered a working solution on time by focusing on core requirements first. The experimental features were gradually introduced in subsequent releases, leading to a 40% improvement in accuracy.

8. Describe a situation where you had to balance model accuracy with deployment constraints.

Situation: A client needed real-time object detection on edge devices with limited computing power.

Task: I had to optimize a large model (EfficientDet-D7) to run on devices with 2GB RAM while maintaining acceptable accuracy.

Action: Implemented model quantization, pruning, and knowledge distillation. Created a custom architecture that focused on detecting specific object classes instead of general detection.

Result: The optimized model ran 5x faster, used 75% less memory, and maintained 90% of the original accuracy. The solution was successfully deployed to over 1000 edge devices.

9. Tell me about a time when you had to debug a complex computer vision pipeline in production.

Situation: Our production system started showing degraded performance in specific lighting conditions.

Task: I needed to identify and fix the issue while minimizing system downtime.

Action: I implemented detailed logging at each pipeline stage, created visualization tools for intermediate results, and used A/B testing to isolate the problem. Discovered that recent changes to image normalization were causing issues.

Result: Fixed the issue within 24 hours, implemented automated testing for various lighting conditions, and created documentation for similar debugging scenarios.

10. How do you approach knowledge sharing and documentation in your computer vision projects?

Situation: Our team was growing rapidly, and new members were struggling to understand complex CV pipelines.

Task: I needed to improve knowledge transfer and reduce onboarding time.

Action: Created a comprehensive wiki with architecture diagrams, example notebooks, and common debugging scenarios. Organized bi-weekly technical sessions for deep dives into specific components. Implemented automated documentation generation from code.

Result: Reduced new developer onboarding time from 4 weeks to 2 weeks. Improved code quality as developers better understood best practices and existing implementations.