High-Accuracy Micro-Defect Detection System

A state-of-the-art computer vision pipeline for detecting micro-defects ("blisters") on mobile phone backglass with >99% recall and >95% precision.

Overview

This project implements a **Three-Stage Refinement Pipeline** that progressively narrows focus from raw image to precise defect detection:

1. **Stage 1**: ROI Isolation - Find and align the phone backglass
2. **Stage 2**: Candidate Detection - High-recall search for all potential defects
3. **Stage 3**: Verification - High-precision filtering and final segmentation

Architecture

Stage 1: Robust ROI Isolation

- **Model**: YOLOv8-Small + Segment Anything Model (SAM)
- **Purpose**: Extract perfectly aligned backglass region
- **Output**: Standardized, rectangular backglass image

Stage 2: High-Recall Candidate Search

- **Model**: YOLOv8-Large-Seg with P2 head (stride-4 detection)
- **Strategy**: Adaptive SAHI (Slicing Aided Hyper Inference)
- **Purpose**: Find every possible defect candidate
- **Output**: List of candidates with high recall, some false positives

Stage 3: High-Precision Verification

- **Model**: Lightweight U-Net Refiner
- **Purpose**: Filter false positives, generate clean masks
- **Output**: Final defect list with bounding boxes and segmentation masks

Project Phases

Phase 1: Data Foundation (Weeks 1-4)

- **Environment Setup**: GPU server with PyTorch, OpenCV, CUDA
- **Image Acquisition**: 2,000+ high-resolution images
- **Annotation**: Pixel-perfect segmentation masks for all defects
- **Augmentation Library**: 300+ defect patches for copy-paste augmentation

Phase 2: Model Development (Weeks 5-8)

- **Stage 1 Training**: YOLOv8-Small on ROI bounding boxes
- **Stage 2 Training**: P2-enabled YOLOv8-Large-Seg with heavy augmentation
- **Stage 3 Training**: U-Net on balanced true/false positive crops

Phase 3: Integration & Validation (Weeks 9-10)

- **Pipeline Integration**: End-to-end inference script
- **Performance Benchmarking**: Validation on held-out test set
- **Error Analysis**: Systematic failure case analysis

Phase 4: Finalization (Weeks 11-12)

- **Iterative Improvement**: Retrain based on error analysis
- **Documentation**: Complete technical documentation
- **Demo Application**: Interactive visualization tool

Technical Specifications

Hardware Requirements

- **GPU**: NVIDIA A6000/RTX 4090 (24GB+ VRAM recommended)
- **RAM**: 32GB+ system memory
- **Storage**: 500GB+ for datasets and models

Software Dependencies

```
# Core ML Framework
torch>=2.0.0
torchvision>=0.15.0
ultralytics>=8.0.0
```

```
# Computer Vision
opencv-python>=4.8.0
segment-anything>=1.0

# Data Processing
numpy>=1.24.0
pandas>=2.0.0
pillow>=10.0.0

# Visualization
matplotlib>=3.7.0
seaborn>=0.12.0
```

Model Configurations

Stage 1: ROI Detector

```yaml
 # Training Parameters
imgsz: 640
batch: 16
epochs: 50
lr0: 0.001
optimizer: AdamW
patience: 10
```

Stage 2: Candidate Detector

```yaml
 # Training Parameters
imgsz: 1280
batch: 4
epochs: 150
lr0: 0.001
weight_decay: 0.0005

# Augmentation
mosaic: 1.0
fliplr: 0.5
flipud: 0.5
copy_paste: 0.5

# Loss Weights
box: 7.5
cls: 0.5
dfl: 1.5
fl_gamma: 1.5
```

Stage 3: Refiner U-Net

```yaml
# Architecture
input_size: 128
channels: [16, 32, 64]
depth: 3

# Training
batch_size: 64
epochs: 75
lr: 3e-4
optimizer: Adam

# Loss Function
loss: 0.7 * DiceLoss + 0.3 * BCELoss
```

Performance Targets

Metric	Target	Description
Defect Recall	>99.0%	Percentage of true defects detected
Defect Precision	>95.0%	Percentage of detections that are real defects
Mask IoU	>0.70	Segmentation mask accuracy
ROI Success Rate	>99.9%	Backglass isolation success rate

Usage

Training Pipeline

```bash
# Stage 1: Train ROI detector
python train_roi.py --data roi_dataset.yaml --epochs 50

# Stage 2: Train candidate detector
python train_candidates.py --data defect_dataset.yaml --epochs 150

# Stage 3: Train refiner
python train_refiner.py --data refiner_dataset.yaml --epochs 75
```

Inference Pipeline

```
# Run complete detection pipeline
python detect.py --image path/to/image.jpg --output results.json

# Batch processing
python detect.py --folder path/to/images/ --output results/
```

Output Format

```json
{
  "image": "sample.jpg",
  "defects": [
    {
      "id": 1,
      "bbox": [x, y, width, height],
      "mask": "base64_encoded_mask",
      "confidence": 0.95,
      "area_pixels": 156
    }
  ],
  "total_defects": 1,
  "processing_time": 1.2
}
```

Key Innovations

P2 Feature Head

- Enables detection at stride-4 for objects as small as 8-16 pixels
- Critical architectural modification for micro-defect detection

Adaptive SAHI Strategy

- Dynamic tile sizing based on defect probability
- Focuses computational resources on high-likelihood regions

Specialized Refiner Network

- Trained exclusively on hard negatives from Stage 2
- Acts as expert classifier for final verification

Copy-Paste Augmentation

- Library of 300+ defect patches with transparent backgrounds
- Realistic blending with Poisson editing techniques

Data Requirements

Image Collection

- **Quantity**: 2,000+ high-resolution images
- **Variety**: All defect types, sizes, locations
- **Quality**: Consistent lighting, sharp focus
- **Negatives**: Clean units + confusing artifacts (dust, fibers)

Annotation Standards

- **Tool**: CVAT or Labelbox recommended
- **Format**: COCO or YOLO segmentation format
- **Quality**: Pixel-perfect masks, consistent labeling
- **Validation**: Multi-annotator agreement >95%

Deployment Considerations

Inference Optimization

- **Model Quantization**: INT8 quantization for production
- **TensorRT**: GPU acceleration for real-time processing
- **Batch Processing**: Optimize for throughput vs latency

Quality Assurance

- **Active Learning**: Continuous model improvement loop
- **Hard Negative Mining**: Systematic false positive collection
- **Performance Monitoring**: Real-time accuracy tracking

Troubleshooting

Common Issues

1. **Low Recall**: Check P2 head implementation, increase augmentation

2. **High False Positives**: Retrain Stage 3 with more hard negatives
3. **ROI Failures**: Improve Stage 1 dataset diversity
4. **Memory Issues**: Reduce batch size, use gradient accumulation

Performance Optimization

- Use mixed precision training (FP16)
- Implement gradient checkpointing for large models
- Optimize data loading with multiple workers

Contributing

Development Workflow

1. Fork repository and create feature branch
2. Implement changes with comprehensive tests
3. Update documentation and performance benchmarks
4. Submit pull request with detailed description

Code Standards

- Follow PEP 8 style guidelines
- Include docstrings for all functions
- Add unit tests for new functionality
- Maintain >90% code coverage

License

Contact

For technical questions or implementation support, please contact the development team.

Note: This system represents cutting-edge computer vision technology specifically designed for industrial quality control applications. The three-stage architecture ensures maximum accuracy while maintaining practical deployment feasibility.