Documentação Técnica - Sistema de Visão Computacional FPF

Versão: 1.0.0
Data: 31 de Julho de 2025
Autores: Diógeles da Silva Tamaturgo

.

1. Visão Geral da Solução

1.1 Propósito

O Sistema de Visão Computacional FPF foi desenvolvido para automatizar a detecção de objetos industriais e extração de informações de QR codes em ambientes de armazém. A solução processa imagens de forma assíncrona, identificando paletes, caixas, empilhadeiras e decodificando QR codes com alta precisão.

1.2 Principais Funcionalidades

- **Detecção Multi-Classe**: Identificação simultânea de 4 classes de objetos (pallet, box, forklift, qr_code)
- **Processamento Assíncrono**: Utilizando Celery para processamento em background
- **Decodificação Robusta de QR**: Pipeline com 7 estratégias diferentes para maximizar taxa de sucesso
- **API RESTful**: Interface padronizada com documentação automática
- **Cache Inteligente**: Sistema de cache multi-camada (Redis + PostgreSQL)
- **Escalabilidade Horizontal**: Arquitetura distribuída pronta para múltiplos workers

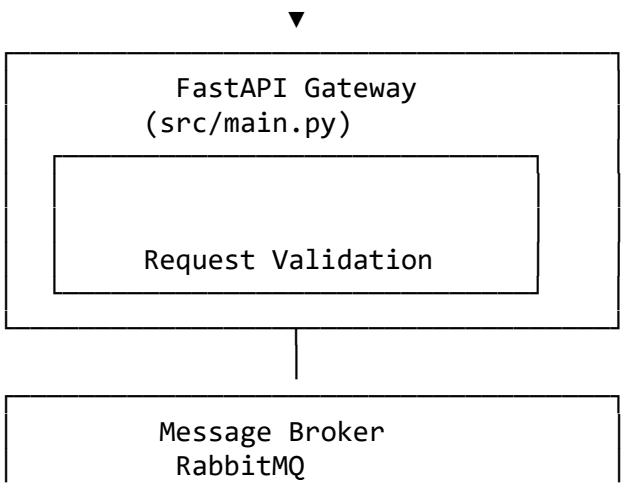
1.3 Benefícios de Negócio

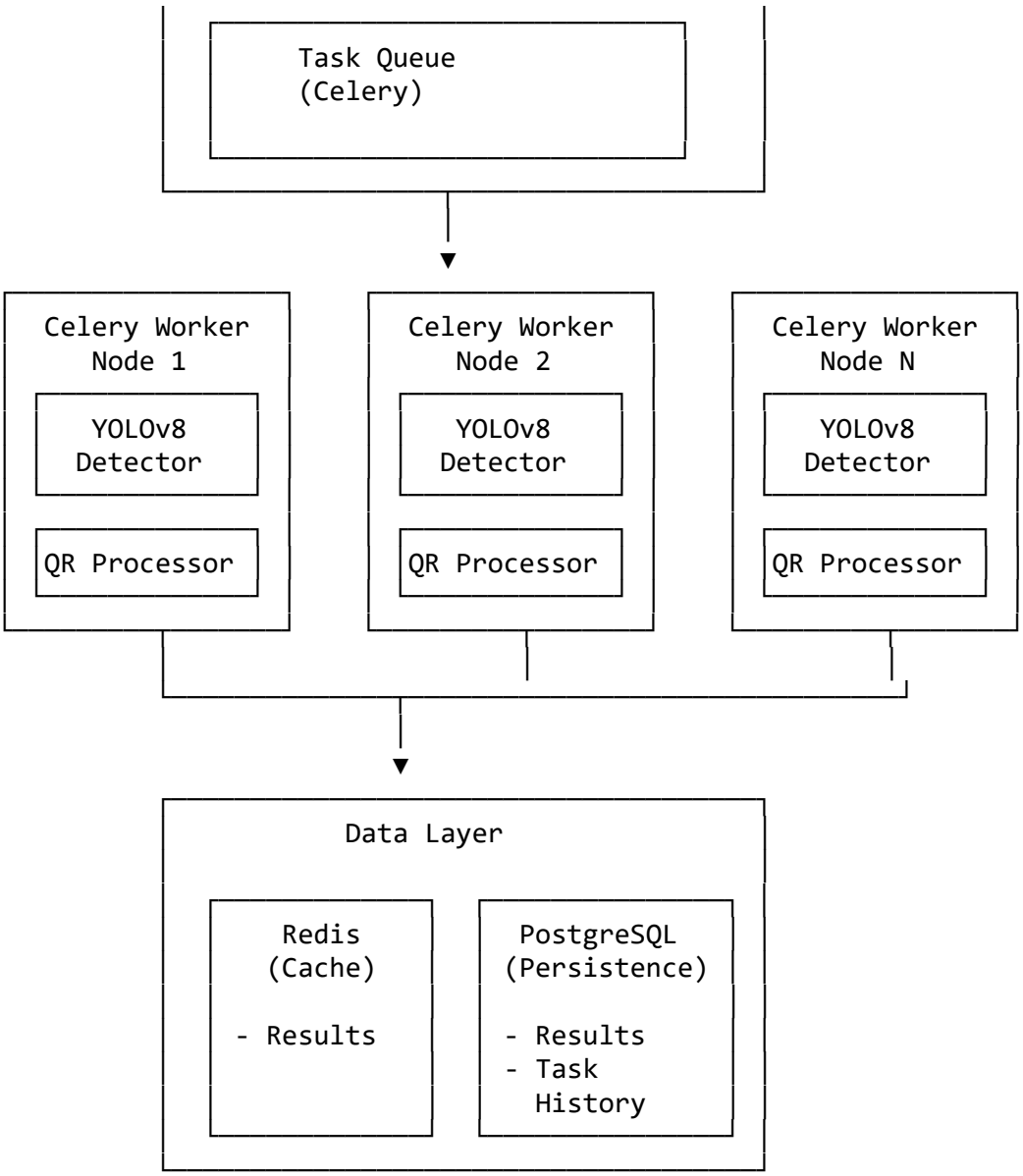
- **Precisão**: Taxa de detecção superior a 97.6% (mAP@0.5)
- **Escalabilidade**: Capacidade de processar centenas de imagens simultaneamente
- **Rastreabilidade**: Histórico completo de processamentos para auditoria

.

2. Arquitetura do Sistema

2.1 Diagrama de Arquitetura





2.2 Padrões Arquiteturais

Microserviços: Cada componente tem responsabilidade única e bem definida

Event-Driven: Comunicação assíncrona via mensagens

CQRS: Separação entre operações de comando e consulta

Circuit Breaker: Proteção contra falhas em cascata

Retry Pattern: Reprocessamento automático em caso de falhas temporárias

.

3. Stack Tecnológico

3.1 Backend

Tecnologia	Versão	Propósito
Python	3.11+	Linguagem principal
FastAPI	0.104.1	Framework Web

Tecnologia	Versão	Propósito
Celery	5.3.4	Task Queue
Pydantic	2.4.2	Validação de dados

3.2 Computer Vision & ML

Tecnologia	Versão	Propósito
YOLOv8	8.0.196	Detecção de objetos
OpenCV	4.8.1.78	Processamento de imagem
NumPy	2.3.2	Operações matriciais
Pillow	10.0.1	Manipulação de imagens
PyZBar	0.1.9	Decodificação QR

3.3 Infraestrutura

Tecnologia	Versão	Propósito
PostgreSQL	15-alpine	Banco de dados
Redis	7-alpine	Cache/Result Backend
RabbitMQ	3-management	Message Broker
Docker	Latest	Containerização

.

4. Decisões Técnicas

4.1 FastAPI vs Alternativas

Decisão: FastAPI
Alternativas Consideradas: Flask,

Justificativas: - **Async Native**: Suporte nativo a programação assíncrona - **Documentação Automática**: OpenAPI/Swagger integrado - **Validação Automática**: Pydantic integrado reduz código boilerplate - **Type Hints**: Suporte completo a type hints do Python

4.2 YOLOv8 vs Alternativas

Decisão: YOLOv8
Alternativas Consideradas: YOLOv11

Justificativas: - **Precisão**: mAP@0.5 de 0.847 no dataset customizado - **Facilidade de Treinamento**: Pipeline simplificado da Ultralytics - **Suporte**: Comunidade ativa e documentação robusta - **Flexibilidade**: Múltiplos modelos (nano, small, medium, large)

4.3 Celery vs Alternativas

Decisão: Celery
Alternativas Consideradas: RQ, Dramatiq, Apache Airflow

Justificativas: - **Escalabilidade**: Suporte nativo a múltiplos workers e brokers - **Flexibilidade**: Suporte a múltiplos backends e brokers - **Comunidade**: Vasta documentação e casos de uso

4.4 PostgreSQL vs Alternativas

Decisão: PostgreSQL
Alternativas Consideradas: MongoDB, MySQL, SQLite

Justificativas: - **ACID Compliance**: Garantia de consistência de dados - **JSON Support**: Campos JSON nativos para metadados flexíveis - **Performance**: Otimizações avançadas para queries complexas - **Extensibilidade**: Suporte a extensões como PostGIS (futuro) - **Confiabilidade**: Reputação sólida em ambientes empresariais

4.5 Estratégia de QR Code Processing

Decisão: Pipeline Multi-Estratégia
Alternativas Consideradas: Single Strategy, External Service

Justificativas: - **Robustez**: Funciona com QR codes danificados, rotacionados ou com baixa qualidade - **Controle**: Implementação interna permite otimizações específicas, além de garantir o desacoplamento da lógica de negócio - **Latência**: Processamento local evita chamadas de rede

.

5. Componentes do Sistema

5.1 API Gateway (FastAPI)

Localização: `src/main.py`
Responsabilidades: - Recepção e validação de requisições HTTP - Autenticação e autorização - Rate limiting e throttling - Roteamento para controllers apropriados - Geração de documentação automática

Configurações Principais:

```
app = FastAPI(
    title="FPF Vision API - Computer Vision Processing",
    description="API para processamento assíncrono de imagens",
    version="1.0.0",
    docs_url="/docs",
)
```

5.2 Controllers Layer

Localização: `src/api/controllers/`
Padrão: MVC Controller Pattern

ImageController (`image_controller.py`): - Coordena upload e validação de imagens - Cria tasks Celery para processamento - Gerencia consulta de resultados - Implementa cache strategy

5.3 Services Layer

Localização: `src/api/services/`
Padrão: Service Layer Pattern

ImageService: - Lógica de negócio para processamento de imagens - Orquestração entre detecção YOLO e QR processing - Formatação de resultados - Gerenciamento de armazenamento

5.4 Tasks Layer (Celery Workers)

Localização: `src/api/tasks/`
Padrão: Task Queue Pattern

ImageProcessingTasks:

```python
@celery_app.task(bind=True, name="process_image")
def process_image_task(self, image_path: str, task_id: str):
    # Processamento assíncrono da imagem
    pass
```

5.5 Core Processing Engine

Localização: `src/core/`

YoloDetector (`detection/yolo_detector.py`): - Carregamento e inicialização do modelo YOLOv8 - Execução de inferência com otimizações - Post-processing de detecções

QRDecoder (`processing/qr_decoder.py`): - Pipeline de 7 estratégias de decodificação - Pré-processamento de imagens para QR - Validação e formatação de resultados

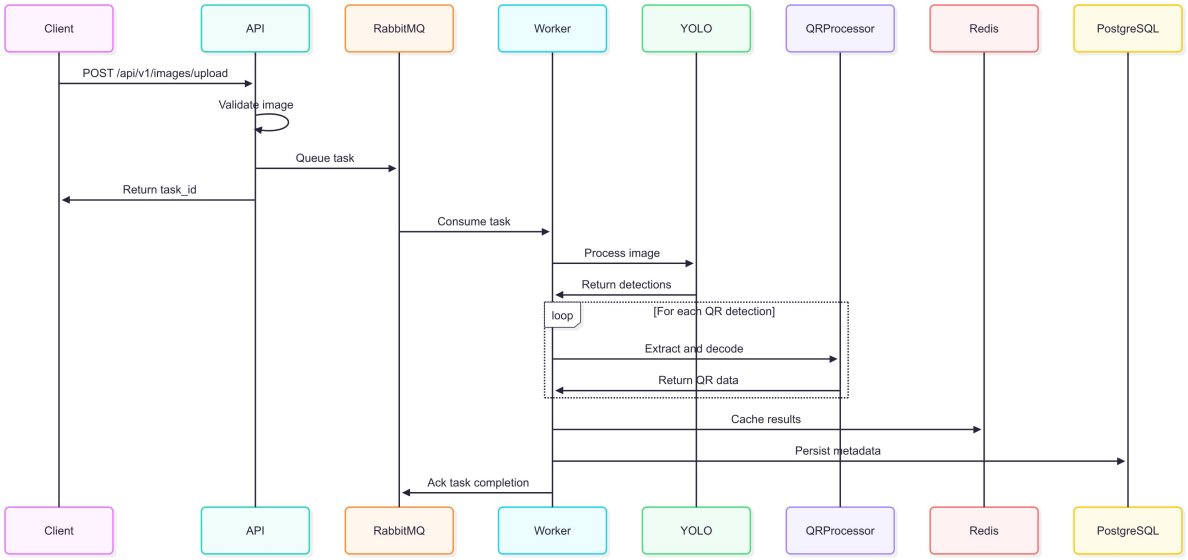
5.6 Data Access Layer

Localização: `src/db/`
Padrão: Repository Pattern

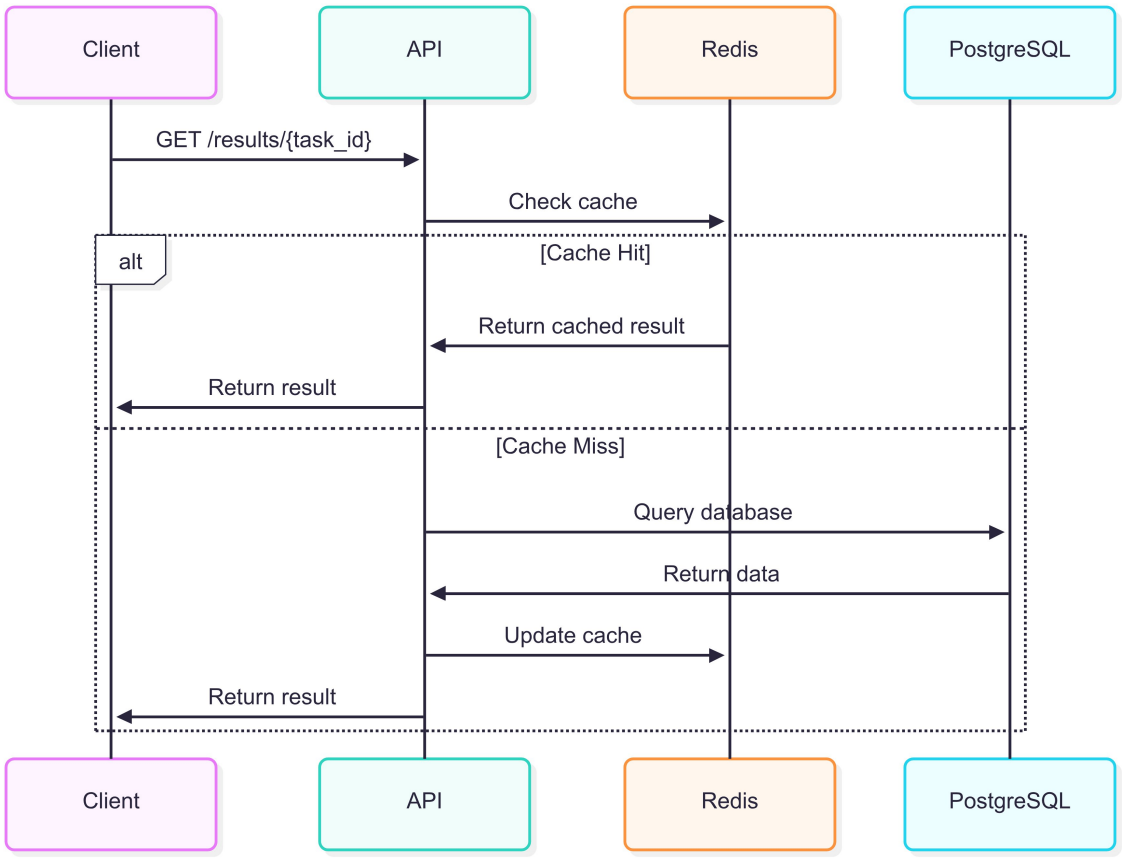
Models: - `TaskResult`: Metadados de processamento - `DetectedObject`: Objetos detectados - `QRCode`: Códigos QR decodificados

6. Fluxo de Dados

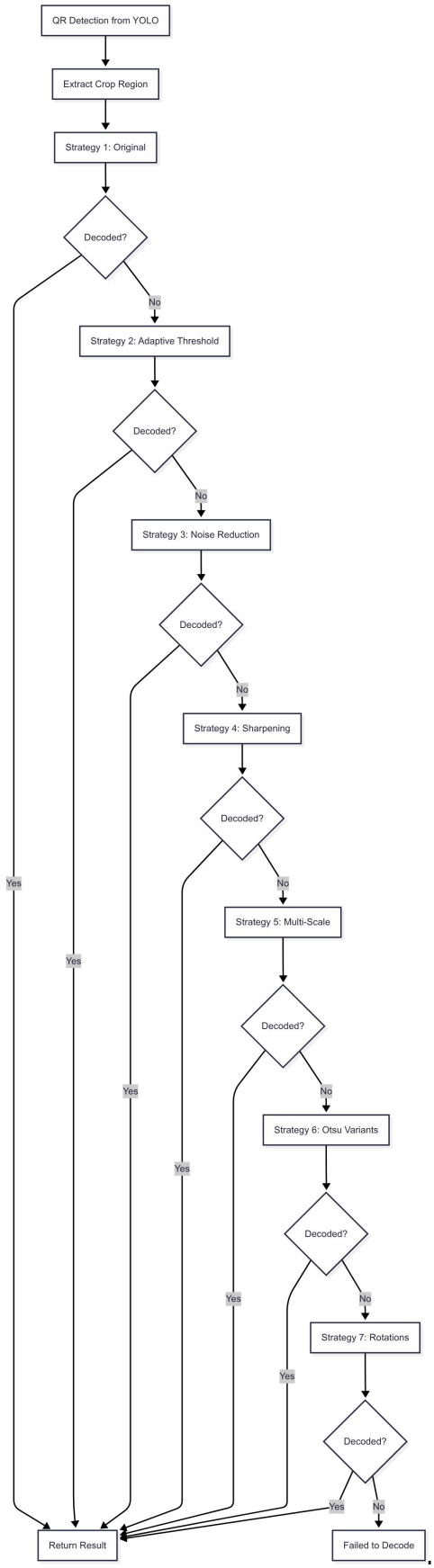
6.1 Fluxo de Upload e Processamento



6.2 Fluxo de Consulta de Resultados



6.3 Pipeline de Processamento de QR



7. API e Endpoints

7.1 Estrutura da API

Base URL: `http://localhost:8080/api/v1`
Documentação: `http://localhost:8080/docs`

7.2 Endpoints Principais

7.2.1 Upload de Imagem

Endpoint: `POST /images/upload`
Content-Type: `multipart/form-data`

Request:

```
curl -X POST "http://localhost:8080/api/v1/images/upload" \
     -H "Content-Type: multipart/form-data" \
     -F "file=@warehouse_image.jpg"
```

Response:

```
{
  "task_id": "abc123-def456-ghi789",
  "status": "queued",
  "message": "Image uploaded successfully and queued for processing",
  "estimated_processing_time": "30-60 seconds"
}
```

7.2.2 Upload Múltiplo

Endpoint: `POST /images/upload-multiple`

Request:

```
curl -X POST "http://localhost:8080/api/v1/images/upload-multiple" \
     -H "Content-Type: multipart/form-data" \
     -F "files=@image1.jpg" \
     -F "files=@image2.jpg" \
     -F "files=@image3.jpg"
```

Response:

```
[
  {
    "task_id": "task_001",
    "status": "queued",
    "message": "Image uploaded successfully"
  },
  {
    "task_id": "task_002",
    "status": "queued",
    "message": "Image uploaded successfully"
  }
]
```

7.2.3 Consulta de Resultado

Endpoint: `GET /results/{task_id}`

Request:

```
curl -X GET "http://localhost:8080/api/v1/results/abc123-def456-ghi789"
```

Response Completa:


```
{
  "task_id": "abc123-def456-ghi789",
  "status": "completed",
  "processing_time": 1.23,
  "image_info": {
    "filename": "warehouse_001.jpg",
    "dimensions": {
      "width": 1920,
      "height": 1080
    },
    "size_bytes": 245760
  },
  "detected_objects": [
    {
      "object_id": "OBJ_001",
      "class": "pallet",
      "confidence": 0.92,
      "bounding_box": {
        "x": 150,
        "y": 200,
        "width": 300,
        "height": 180,
        "normalized": {
          "x": 0.078,
          "y": 0.185,
          "width": 0.156,
          "height": 0.167
        }
      }
    },
    {
      "object_id": "OBJ_002",
      "class": "box",
      "confidence": 0.87,
      "bounding_box": {
        "x": 500,
        "y": 300,
        "width": 120,
        "height": 90
      }
    },
    {
      "object_id": "OBJ_003",
      "class": "forklift",
      "confidence": 0.94,
      "bounding_box": {
        "x": 800,
        "y": 400,
        "width": 200,
        "height": 150
      }
```

```
    }
  ],
  "qr_codes": [
    {
      "qr_id": "QR_001",
      "data": "PALLET-ABC-123-WAREHOUSE-ZONE-A",
      "confidence": 0.98,
      "position": {
        "x": 200,
        "y": 150,
        "width": 80,
        "height": 80
      },
      "decoding_strategy": "adaptive_threshold",
      "raw_bytes": "80 bytes"
    },
    {
      "qr_id": "QR_002",
      "data": "BOX-DEF-456-LOT-2024-07",
      "confidence": 0.95,
      "position": {
        "x": 520,
        "y": 320,
        "width": 60,
        "height": 60
      },
      "decoding_strategy": "original"
    }
  ],
  "statistics": {
    "total_objects": 3,
    "total_qr_codes": 2,
    "detection_confidence_avg": 0.91,
    "qr_success_rate": 1.0
  },
  "metadata": {
    "processed_at": "2025-07-31T14:30:45Z",
    "worker_id": "worker_001",
    "model_version": "yolov8n_custom_v1.0",
    "processing_config": {
      "confidence_threshold": 0.47,
      "enable_qr_detection": true
    }
  }
}
```


7.2.4 Listagem com Filtros

Endpoint: `GET /results`

Request:

```
curl -X GET
"http://localhost:8080/api/v1/results?page=1&limit=10&status=completed&start_
date=2025-07-01T00:00:00&end_date=2025-07-31T23:59:59"
```

Response:

```
{
  "tasks": [
    {
      "task_id": "task_001",
      "status": "completed",
      "created_at": "2025-07-31T14:30:45Z",
      "processing_time": 1.23,
      "objects_count": 3,
      "qr_codes_count": 2
    }
  ],
  "pagination": {
    "page": 1,
    "limit": 10,
    "total": 157,
    "pages": 16,
    "has_next": true,
    "has_prev": false
  },
  "filters_applied": {
    "status": "completed",
    "date_range": {
      "start": "2025-07-01T00:00:00",
      "end": "2025-07-31T23:59:59"
    }
  }
}
```


7.3 Status da API

7.3.1 Health Check

Endpoint: `GET /api/v1/health`

Response:


```
{
  "status": "healthy",
  "timestamp": "2025-07-31T14:30:45Z",
  "services": {
    "database": "connected",
    "redis": "connected",
    "rabbitmq": "connected",
    "celery_workers": {
      "active": 3,
      "total": 5
    }
  },
  "system": {
    "cpu_usage": "15%",
    "memory_usage": "2.1GB/8GB",
    "disk_usage": "45GB/100GB"
  }
}
```

8. Modelo de Machine Learning

8.1 YOLOv8 - Especificações

Arquitetura: YOLOv8 Nano (yolov8n)
Entrada: 640x640 pixels
Classes: 4 (pallet, box, forklift, qr_code)
Formato: PyTorch (.pt)

8.2 Métricas de Performance

Treinamento: - **Épocas**: 40 - **Batch Size**: 8 - **Learning Rate**: 0.01 (com scheduler) - **Data Augmentation**: Rotação, flip, mudança de escala, ajuste de brilho

Resultados Finais:

Class	Images	Instances	Box(P	R	mAP50	mAP50-95):
all	2933	4924	0.965	0.944	0.976	0.826
box	2933	2619	0.971	0.957	0.982	0.868
qr_code	2933	1003	0.961	0.959	0.98	0.864
pallet	2933	927	0.992	0.992	0.994	0.863
forklift	2933	375	0.935	0.869	0.947	0.709

9. Processamento de QR Codes

9.1 Pipeline Multi-Estratégia

O sistema implementa 7 estratégias sequenciais para maximizar a taxa de decodificação de QR codes, especialmente em condições adversas comuns em ambientes industriais.

9.2 Estratégias Detalhadas

Estratégia 1: Original

Descrição: Decodificação direta da região crop sem processamento
Caso de Uso: QR codes em boa qualidade e contraste adequado

Estratégia 2: Adaptive Threshold

Descrição: Binarização adaptativa com filtro gaussiano
Caso de Uso: QR codes com iluminação irregular

Estratégia 3: Noise Reduction

Descrição: Filtro mediano seguido de threshold Otsu
Caso de Uso: QR codes com ruído ou artifacts de compressão

Estratégia 4: Sharpening

Descrição: Filtro de nitidez para melhorar bordas
Caso de Uso: QR codes desfocados ou com baixa definição

Estratégia 5: Multi-Scale

Descrição: Redimensionamento em múltiplas escalas
Caso de Uso: QR codes muito pequenos ou muito grandes

Estratégia 6: Otsu Variants

Descrição: Múltiplas variações do algoritmo Otsu
Caso de Uso: QR codes com distribuição de pixels não-padrão

Estratégia 7: Rotations

Descrição: Rotações de 90°, 180° e 270°
Caso de Uso: QR codes rotacionados ou mal orientados

9.3 Otimizações Implementadas

9.3.1 Early Stopping

O pipeline para assim que uma estratégia consegue decodificar com sucesso, evitando processamento desnecessário.

9.3.2 Caching de Crops

QR crops são salvos temporariamente para debug e análise posterior:

```
if self.config.get("save_crops", True):
    crop_path = f"{QR_CROPS_DIR}/qr_{task_id}_{qr_id}.jpg"
    cv2.imwrite(crop_path, crop_image).
```

Conclusão

O Sistema de Visão Computacional FPF representa uma solução robusta e escalável para detecção de objetos industriais e processamento de QR codes. A arquitetura baseada em microserviços, processamento assíncrono e pipeline multi-estratégia garante alta disponibilidade, performance otimizada na decodificação de QR codes.