CAPÍTULO 7

CONCLUSÃO, CICLO DE VIDA E IMPLANTAÇÃO

1. SÍNTESE DO CICLO DE DESENVOLVIMENTO

O ciclo de desenvolvimento de detectores baseados na família YOLO (do YOLOv8 ao YOLOv12) integra a modelagem arquitetural, o ajuste fino de hiperparâmetros, a avaliação quantitativa com matrizes de confusão e curvas P-R, o dimensionamento de hardware e as rotinas automatizadas de backup com Rclone para proteger os checkpoints contra quedas de energia.

2. COLETA, ANOTAÇÃO E PRÉ-PROCESSAMENTO DE DADOS

A base de dados define o limite de desempenho do detector. Duas etapas distintas atuam sobre as imagens:

- **Coleta e curadoria de dados reais:** captação de novas imagens sob variações operacionais (ângulos, iluminação e oclusões). A coleta acrescenta padrões visuais inéditos à base.
- **Aumento de dados (*data augmentation*):** transformações geométricas e de cor no treino (rotações, cortes e *mosaic*) para ampliar a diversidade amostral sem novas coletas.

Para a anotação das caixas delimitadoras (*bounding boxes*) com coordenadas normalizadas em arquivos *.txt*, utilizam-se ambientes como:

- **CVAT (*Computer Vision Annotation Tool*):** ferramenta de código aberto para rotulagem de imagens e vídeos em projetos colaborativos.
- **Label Studio:** aplicação configurável para anotação multimodal com integração a modelos pré-treinados de apoio.
- **Roboflow:** serviço para versionamento de bases, padronização de resolução e exportação direta para a estrutura de pastas do YOLO.

3. IMPLANTAÇÃO PRÁTICA: INFERÊNCIA, VALIDAÇÃO E EXPORTAÇÃO

Após a convergência do treinamento, o modelo passa pela validação em partição de teste independente, pela execução de inferências e pela conversão para formatos de execução otimizada. O script a seguir executa esse fluxo com o pacote *ultralytics*:

Script: Validação, Inferência e Exportação de Modelos YOLO

```python
from ultralytics import YOLO

def main():
    # 1. Carregar o modelo treinado (best.pt)
    model = YOLO("runs/detect/treinamento_yolo/weights/best.pt")

    # 2. Validacao formal no conjunto de teste isolado
    print("--- Executando Validacao Formal ---")
    metrics = model.val(data="data.yaml", split="val")
    print(f"mAP50-95 obtido: {metrics.box.map:.4f}")
    print(f"mAP50 obtido: {metrics.box.map50:.4f}")

    # 3. Inferencia em novas imagens ou videos
    print("\n--- Executando Inferencia ---")
    results = model.predict(
        source="dataset/images/val",
        conf=0.25,
        save=True,
        project="runs/detect",
        name="inferencia_teste"
    )
    print("Inferencia concluida. Resultados salvos em runs/detect/inferencia_teste")

    # 4. Exportacao para formatos otimizados de producao
    print("\n--- Exportando Modelo para Producao ---")
    # Exportacao para ONNX (Open Neural Network Exchange)
    onnx_file = model.export(format="onnx", dynamic=True)
    print(f"Modelo exportado para ONNX: {onnx_file}")

    # Exportacao para TensorRT (requer GPU NVIDIA e TensorRT instalado)
    try:
        engine_file = model.export(format="engine", device=0)
        print(f"Modelo exportado para TensorRT: {engine_file}")
    except Exception as e:
        print(f"Aviso: Exportacao TensorRT nao executada ({e})")


if __name__ == "__main__":
    main()
```

4. IMPLANTAÇÃO EM AMBIENTES DE PRODUÇÃO (EDGE E CLOUD)

O formato ONNX permite carregar o modelo via *onnxruntime* em linguagens como C++, C# e Python, sem a necessidade de instalar o PyTorch no servidor ou dispositivo de destino. Para dispositivos de borda (*Edge Computing*), como placas NVIDIA Jetson, Raspberry Pi e processadores dedicados, aplicam-se motores específicos:

- **TensorRT (*.engine*):** compila o grafo com fusão de camadas e quantização em meia precisão (FP16) ou inteiros de 8 bits (INT8) para GPUs NVIDIA.
- **OpenVINO, TFLite e NPUs:** adaptam a rede para arquiteturas Intel x86, núcleos ARM e aceleradores neurais dedicados.

5. CONSIDERAÇÕES FINAIS

A implantação de detectores em condições reais é um processo iterativo. Resultados preliminares frequentemente acusam perdas em iluminação desfavorável, sobreposições de classes com morfologia semelhante ou falsos negativos em objetos de pequena escala. A correção dessas falhas envolve a leitura sistemática da matriz de confusão, o reforço amostral nos cenários de erro, a recalibração de hiperparâmetros de perda e o teste comparativo entre versões de diferentes capacidades da arquitetura.

REFERÊNCIAS

JOCHER, Glenn; CHAURASIA, Ayush; QIU, Jing. Ultralytics YOLO. [S.l.: s.n.], 2023. Software de detecção, segmentação, classificação e estimativa de pose baseado na família YOLO. Disponível em: https://github.com/ultralytics/ultralytics.

ONNX RUNTIME DEVELOPERS. ONNX Runtime: cross-platform, high performance ML inferencing and training accelerator. 2026. Disponível em: https://onnxruntime.ai.