

スケーラブルな商用レベルの物体検出モデル構築：GoogleColabPro+とLibreYOLOを活用した実践アプローチ

物体検出は自律システムやセキュリティ分析など、現代のコンピュータビジョンにおける基盤技術です。しかし、学習フレームワークのエコシステムには制限の厳しいソフトウェアライセンスが存在し、商用デプロイを複雑にすることがあります。本記事では、100%オープンソースの**LibreYOLO**とGoogle Colab Pro+を組み合わせ、制約のない商用グレードの物体検出モデルをゼロから学習・デプロイするワークフローを解説します。

LibreYOLO: 真のオープンソースの選択肢

歴史的に「YOLO」ファミリーはリアルタイム物体検出に革命をもたらしましたが、近年の人気フレームワークはAGPLライセンスを採用しており、プロプライエタリなSaaSやクローズドソースの商用製品に組み込むには高額なエンタープライズライセンスが必要になるという課題がありました。

LibreYOLOはこの問題を解決するために構築された、100% **MITライセンス**のコンピュータビジョンエンジンです。主な利点は以下の通りです。

- **真のオープンソース:** 依存関係チェーンにAGPLコードが一切含まれないため、クローズドソース製品にも安全に統合できます。
- **複数アーキテクチャの統合API:** 単一のAPIでYOLOv9、YOLOX、RF-DETR、RT-DETRなど多様なモデルを学習・検証・デプロイ可能です。
- **End-to-Endのワークフロー:** 組み込みのデータ拡張、混合精度学習、ONNXやTensorRTへのワンラインモデルエクスポートなど、本番デプロイに必要な機能が揃っています。

なぜRF-DETRアーキテクチャを採用したか？

LibreYOLOが提供する豊富なモデル構造の中から、私たちは**RF-DETR**を採用しました。

アンカーボックスやNMS（Non-Maximum Suppression）などの後処理に依存する従来の畳み込みベースのYOLOとは異なり、DETR（DEtection TRansformer）はエンドツーエンドのアプローチを提供します。RF-DETRはVision Transformerの力でグローバルな受容野を持つため、複雑な空間関係の理解、深刻なオクルージョン（物体の重なり）の処理、混雑したシーンでの物体検出に極めて優れています。

Colab Pro+でのコスト効率の高い学習最適化

大規模なTransformerベースのモデル学習は計算負荷が高いですが、Google Colab Pro+の環境特性に合わせて以下のインフラ・学習戦略を最適化しました。

- **プレミアムGPUとコンピュータユニットの活用:** Colab Pro+で利用可能な高性能GPU（A100やV100等）を活用し、限られたコンピュータユニット内で最大のパフォーマンスを引き出します。
- **Google Driveへのアーティファクト保存:** Colabのセッション切断によるデータ消失リスクを防ぐため、データセット、チェックポイント、ログの出力先をマウントしたGoogle Driveへ直接ルーティングします。
- **ローカルストレージ（`/content/`）への展開によるI/O最適化:** Google Driveから数万枚の画像を直接読み込むと深刻なI/Oボトルネックが発生します。これを回避するため、学習開始前にZIP化されたデータセットをColabのローカルストレージにコピー・解凍する処理をパイプラインに組み込みました。

- **早期終了（Early Stopping）の徹底:** 検証メトリクスを監視し、20〜30エポック連続でパフォーマンスが頭打ちになった場合は学習を早期終了させ、GPUリソースとコンピュータユニットを節約します。
- **混合精度学習（FP16）:** LibreYOLO内蔵のAMPサポートを活用することでGPUメモリの消費を半減させ、バッチサイズを最大化しました。

本番レベルの精度の達成

モデルの検証結果は初期ベンチマークを大きく上回り、実環境へのデプロイ準備が整っていることが証明されました。

- **圧倒的なクラス別精度:** 覆面、ナンバープレート、携帯電話といった視覚的変化の少ない物体では98%〜99.8%の精度（mAP@50）を記録しました。また、ナイフや銃といった様々な形状・角度を持つ複雑なセキュリティ上の脅威に対しても95%以上の精度を達成しました。
- **信頼度しきい値の自動キャリブレーション:** 固定のしきい値での運用は本番環境におけるよくある落とし穴です。対象オブジェクトごとに最適な検出しきい値は異なるため、しきい値のスweepを自動化し、F1スコアのカーブの頂点（誤報の最小化と検出感度の最大化のバランス点）を特定する仕組みを構築しました。
- **SAHIによる小物体検出の強化:** 広角カメラの映像では、ナイフや携帯電話などが非常に小さく映ります。高解像度の入力フレームをオーバーラップする小さなタイルに分割して推論し、結果を結合するSAHI（Slicing Aided Hyper Inference）を推論フェーズに統合することで、ベースモデルの構造を変更することなく小物体検出能力を劇的に向上させました。

Python

```
# Colab上でのSAHI推論例
def sahi_predict(model, image, tile_size=640, overlap=0.2, conf=0.25, iou=0.45):
    H, W = image.shape[:2]
    step = int(tile_size * (1 - overlap)) # タイル間のストライド
    boxes = []

    for y in range(0, H - tile_size + 1, step):
        for x in range(0, W - tile_size + 1, step):
            # フルフレームからタイルをクロップ
            tile = image[y:y+tile_size, x:x+tile_size]
            results = model.predict(tile, conf=conf)

            if results.bboxes is None:
                continue

            # フル画像上の座標へリマップ
            for box in results.bboxes.xyxy:
                x1, y1, x2, y2 = box.tolist()
                boxes.append([x1+x, y1+y, x2+x, y2+y])

    # タイル間で重複する検出結果をマージ
    return non_max_suppression(boxes, iou_threshold=iou)
```

データ調達のハードルと前処理

単一のパッケージ化されたデータセットに依存することは現実的ではなかったため、以下の3つの課題に対処しました。

- **マルチソースデータの統合:** Roboflow、Kaggle、GitHubなどから画像を集め、アノテーションの変換（COCO JSONからYOLOフォーマットへ）や重複排除を行う自動パイプラインを構築しました。
- **特化型アングルのキュレーション:** 公開データセットは目線の高さに偏っています。実際のCCTVのような見下ろし型・高所からの視点データを収集することは困難でしたが、現実世界のオーバーヘッドカメラへの汎用性を確保するために不可欠でした。
- **合成データに代わる強力なAugmentation:** AI生成画像はコストが高くリアリズムに欠けるため、Albumentationsによる堅牢なデータ拡張をパイプラインに直接実装しました。ランダムな明るさ調整、ガウシアンブラー、カメラアングルの変更（Perspective）などを適用し、照明や環境の多様性に対する回復力を向上させています。

エンジニアリングの課題と教訓

このColabベースのエンドツーエンドなパイプラインを構築するにあたり、いくつかの重要なハードルを乗り越えました。

- **制限的なフレームワークライセンスの回避:** 当初最大の壁は適切なフレームワークを見つけることでした。多くの物体検出ライブラリは「オープンソース」を謳いながらAGPL等を採用しています[cite: 1]。LibreYOLOの発見により、商用デプロイを妨げない真の許容ライセンスで最新モデルへアクセスできました[cite: 1]。
- **Colabでの動的な環境構築:** フルカスタマイズされた仮想マシンのイメージ（ゴールデンイメージ）を用意できないColab環境では、ライブラリのバージョン不一致によるエラーがサイレントに発生するリスクがあります。そのため、セッション開始時にライブラリのバージョンを厳密に固定してインストールするセットアップセルを用意し、環境の一貫性を担保しました。
- **データセットと現実世界のギャップ:** テストデータでの高い精度が、実際のCCTV環境での成功を保証するわけではありません[cite: 1]。モーションブラーや悪照明など現実環境での課題を克服するためには、継続的に困難なサンプルを収集し、モデルを再学習させるループが不可欠でした[cite: 1]。