Sommaire

Liste des figures et tableaux

Chapitre I Analyse de l'article scientifique "You Only Look Once :Unified, Real-Time Object Detection" de Joseph Redmon, Santosh Divvala, Ross Girshick et Ali Farhadi

I Introduction

Cet article présente YOLO comme une approche unifiée pour la détection d'objets, reformulant le problème comme une régression directe des bounding boxes et probabilités de classes à partir d'images complètes. Contrairement aux méthodes antérieures (ex. classificateurs repurposés comme R-CNN), YOLO utilise un réseau neuronal unique optimisé end-to-end pour la performance de détection. Les auteurs soulignent la vitesse : 45 FPS pour le modèle base, 155 FPS pour Fast YOLO, avec un mAP double des détecteurs temps réel existants. YOLO commet plus d'erreurs de localisation mais moins de faux positifs sur le fond, et généralise mieux à d'autres domaines (ex. artwork).

II Système de détection YOLO

Le système de prédiction de YOLO est résumé en trois grandes étapes, comme l'illustre la figure suivante :

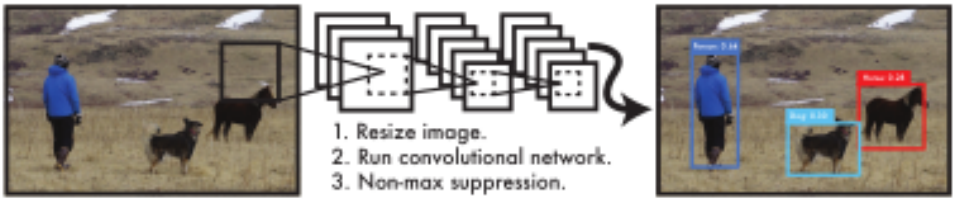

FIGURE 1 – Modèle YOLO [1]

1. Division de l'image en grille
2. Prédiction simultanée par chaque cellule de chaque grille
3. Post-traitement avec suppression non-maximale (NMS)

III Division de l'image en grille (Grid Division)

L'algorithme YOLO divise l'image d'entrée en une grille régulière de taille $\mathbf{S} \times \mathbf{S}$ comme le montre la figure suivante :

- Chaque cellule de la grille est responsable de la détection des objets dont le **centre** tombe à l'intérieur de cette cellule.
- Si le centre d'un objet se trouve dans une cellule, cette cellule doit prédire :
  - une ou plusieurs boîtes englobantes (bounding boxes),
  - la classe de l'objet.
- Si aucun centre d'objet n'est présent dans la cellule, elle prédit simplement « pas d'objet ».

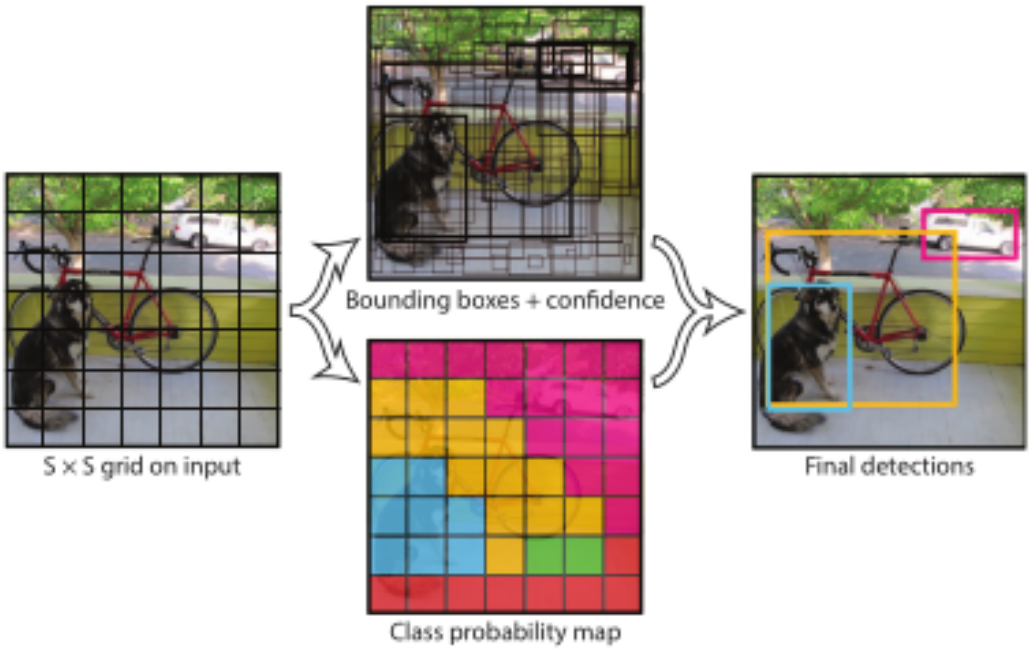

FIGURE 2 – Système de détection YOLO [1]

Cette division permet à YOLO de traiter l'image entière en une seule passe (one-shot), contrairement aux méthodes qui scannent l'image avec des fenêtres glissantes ou des propositions de régions.

- **Avantage** : Raisonnement global sur toute l'image → moins d'erreurs sur le fond.
- **Limitation** : Les objets très proches ou très petits peuvent être mal détectés.

IV Prédiction des boîtes englobantes (Bounding Box Prediction)

Chaque cellule de la grille prédit **B boîtes englobantes**.

Chaque boîte est définie par **5 valeurs** :

- **x, y** : coordonnées du **centre** de la boîte, relatives à la cellule (normalisées entre 0 et 1)
- **w, h** : largeur et hauteur de la boîte, relatives à la taille totale de l'image (normalisées entre 0 et 1)
- **Confidence** : score de confiance = $\Pr(\text{Object}) \times \text{IOU}_{\text{pred}}^{\text{truth}}$

La **confidence** mesure :

- la probabilité qu'il y ait un objet dans la boîte,
- la précision de la prédiction (via l'Intersection over Union avec la vérité terrain).

V Prédiction des classes (Class Prediction)

Chaque cellule prédit **C probabilités conditionnelles** de classes : $\Pr(\text{Class}_i \mid \text{Object})$

— Ces probabilités sont conditionnées par la présence d'un objet dans la cellule.
— Une seule série de probabilités est prédite par cellule (quel que soit le nombre de boîtes B).

Au moment de l'inférence, le score final pour une classe est calculé comme suit :

$$\Pr(\text{Class}_i) = \Pr(\text{Class}_i \mid \text{Object}) \times \Pr(\text{Object}) \times \text{IOU}^{\text{truth}}_{\text{pred}} \tag{1}$$

Ce score combine la probabilité que l'objet appartienne à la classe et la qualité de la boîte.

VI Fonction de perte (Loss Function)

YOLO optimise une **fonction de perte multi-parties** basée sur l'erreur quadratique (sum-squared error). Cette fonction est composée de cinq termes distincts, chacun correspondant à un objectif d'apprentissage différent.

La fonction complète est la suivante :

$$\begin{aligned}
\text{Loss} = {} & \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \\
& + \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2 \right] \\
& + \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} (C_i - \hat{C}_i)^2 \\
& + \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} (C_i - \hat{C}_i)^2 \\
& + \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2
\end{aligned} \tag{2}$$

VI.1 Explication détaillée de chaque terme

1. **Erreur de localisation sur les coordonnées du centre (x, y)**

$$\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right]$$

   — Calcule l'erreur quadratique (MSE) entre le centre prédit $(\hat{x}_i, \hat{y}_i)$ et le centre réel (x_i, y_i).
   — x et y sont relatifs à la cellule de la grille (normalisés entre 0 et 1).

2. **Erreur de localisation sur la taille de la boîte (w, h)**

$$\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2 \right]$$

   — Même principe, mais pour la largeur (w) et la hauteur (h).

— On applique la **racine carrée** avant le calcul de l'erreur.
— **Raison de la racine carrée** : une erreur de 10 pixels sur une petite boîte est beaucoup plus grave que sur une grande boîte. La racine carrée rend l'erreur proportionnellement plus sévère pour les petites tailles.

3. **Erreur de confiance pour les boîtes contenant un objet**

$$\sum_{i=0}^{S^2}\sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} (C_i - \hat{C}_i)^2$$

— Erreur quadratique sur la **confidence** prédite ($\hat{C}_i$) par rapport à la vraie confidence ($C_i = \text{IOU}_{\text{pred}}^{\text{truth}}$).
— Objectif : apprendre que, quand il y a un objet, la confidence doit être élevée (proche de 1 ou de l'IoU réel).

4. **Erreur de confiance pour les boîtes sans objet**

$$\lambda_{\text{noobj}} \sum_{i=0}^{S^2}\sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} (C_i - \hat{C}_i)^2$$

— Même erreur de confidence, mais pour les cellules/boîtes **sans objet** ($C_i = 0$).
— Objectif : pénaliser les faux positifs (prédire un objet là où il n'y en a pas).

5. **Erreur de classification (probabilités de classes)**

$$\sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2$$

— Erreur quadratique sur les probabilités conditionnelles $\Pr(\text{Class}_c \mid \text{Objet})$.
— On compare la probabilité prédite ($\hat{p}_i(c)$) à la vraie (1 pour la classe correcte, 0 sinon).
— Utilise MSE pour simplicité (les versions modernes passent à la cross-entropy).

VI.2 Résumé des objectifs de la fonction de perte

— **Localisation** (termes 1 et 2) → apprendre à bien placer et dimensionner les boîtes (fortement pondérée).
— **Confiance avec objet** (terme 3) → apprendre à être confiant quand il y a un objet.
— **Confiance sans objet** (terme 4) → apprendre à ne pas être confiant quand il n'y a rien (pénalité réduite).
— **Classification** (terme 5) → apprendre à identifier correctement la classe de l'objet (uniquement quand objet présent).

Cette conception astucieuse est la raison pour laquelle YOLO parvient à équilibrer vitesse et précision malgré sa simplicité.

VII Suppression Non-Maximale (Non-Maximum Suppression – NMS)

Après l'inférence, plusieurs boîtes peuvent détecter le même objet.

Étapes du NMS :

1. Trier toutes les boîtes détectées par ordre décroissant de score de confiance.
2. Prendre la boîte avec le score le plus élevé.
3. Supprimer toutes les boîtes qui ont un **IoU > seuil** (typiquement 0.5) avec cette boîte.
4. Répéter jusqu'à ce qu'il ne reste plus de boîte.

Résultat : une seule détection finale par objet, sans doublons.

VIII Architecture du réseau

Le réseau YOLO est un **réseau de neurones convolutifs** (CNN) conçu pour être à la fois très rapide et capable de faire toutes les prédictions en une seule passe.

VIII.1 Structure globale

L'image suivante est la représentation visuelle de l'architecture du réseau YOLO.

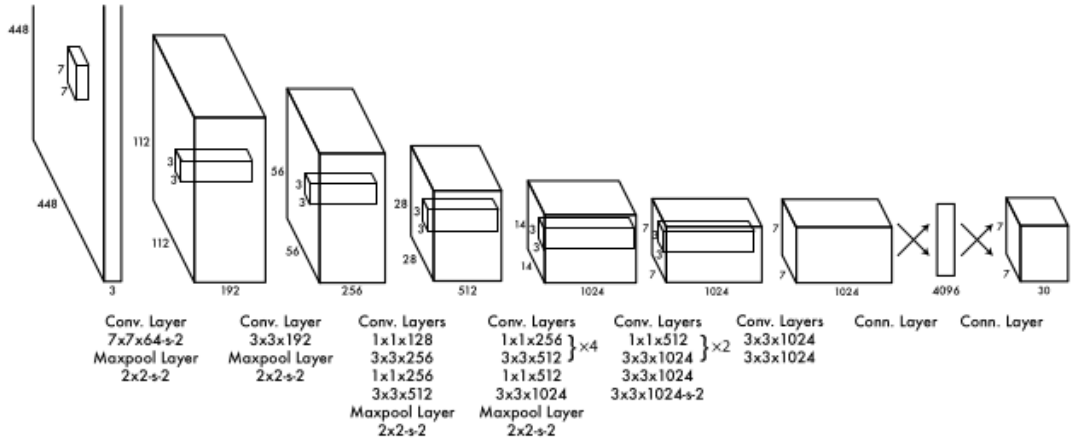

FIGURE 3 – Architecture du réseau YOLO [1]

Le réseau est composé de :

— **24 couches convolutives** → extraction des caractéristiques de l'image
— **2 couches entièrement connectées** (fully connected) → prédictions finales

Il s'inspire fortement de l'architecture **GoogLeNet** (utilisée pour la classification ImageNet), mais simplifiée et adaptée à la détection.

VIII.2 Détail des couches convolutives

Les couches convolutives sont organisées de la façon suivante :

— Les premières couches servent à extraire des caractéristiques de bas niveau (contours, textures, couleurs)
— Les couches intermédiaires détectent des formes plus complexes
— Les dernières couches convolutives produisent des cartes de caractéristiques très riches en information sémantique

Pour réduire la dimensionnalité et accélérer le calcul, YOLO utilise des techniques très efficaces :

- **Couches de réduction 1×1** (bottleneck) → diminuent fortement le nombre de canaux
- **Couches 3×3** → extraient les caractéristiques spatiales

VIII.3 Prétraitement et adaptation pour la détection

- **Pré-entraînement** : les 20 premières couches convolutives sont entraînées sur **ImageNet** (classification) avec une résolution d'entrée de 224×224
- **Adaptation détection** : on ajoute ensuite
  - 4 couches convolutives supplémentaires
  - 2 couches entièrement connectées
- **Changement de résolution** : on passe de 224×224 à **448×448** pour avoir plus de détails fins (important pour la détection d'objets)

VIII.4 Sortie finale du réseau

À la fin du parcours, le réseau produit un tenseur unique de taille :

7 × 7 × 30

Ce tenseur contient **toutes les prédictions** pour toute l'image :

- 7×7 = la grille
- $30 = B\ddot{O}5 + C = 2\ddot{O}5 + 20$ (dans le cas PASCAL VOC)
  - 2 boîtes × 5 valeurs (x, y, w, h, confidence)
  - + 20 probabilités de classes

VIII.5 Version rapide : Fast YOLO

Les auteurs ont aussi créé une version beaucoup plus rapide :

- Seulement **9 couches convolutives** au lieu de 24
- Moins de filtres dans chaque couche
- Même principe d'entraînement et d'inférence
- Vitesse : **155 images par seconde** (contre 45 pour la version de base)

VIII.6 Points clés à retenir

- **Un seul réseau** : de l'image brute jusqu'aux prédictions finales
- **Très profond** : 24 couches convolutives + 2 fully connected
- **Efficace** : utilisation intensive de convolutions 1×1 pour réduire les calculs
- **Pré-entraînement** sur ImageNet → transfert learning
- **Résolution augmentée** : 448×448 au lieu de 224×224 pour plus de précision

C'est cette architecture simple mais puissante qui permet à YOLO d'atteindre des vitesses très élevées tout en restant relativement précis.

Chapitre II Rapport sur le projet de détection de nids de poules sur le tronçon de route Dschang-Bafoussam

I Introduction

Ce rapport résume les étapes que nous avons suivies, du téléchargement du jeu de données à l'entraînement et au déploiement du modèle. Le projet utilise YOLOv8 pour détecter les nids-de-poule à partir de vidéos de caméras dashboard, avec l'objectif d'avertir les conducteurs, créer une carte pour les autorités et quantifier les dommages routiers. Les étapes sont basées sur l'option 2 (datasets publics + affinage local) et des outils comme Kaggle, Roboflow, Ultralytics et Google Colab.

II Choix du Jeu de Données Public et téléchargement

1. **Choix du Dataset :** Nous avons sélectionné le dataset "**Potholes-Detection-YOLOv8**" sur Kaggle (par Angga Dwi Sunarto), qui est déjà au format YOLOv8. Il contient 1581 images pour l'entraînement et 396 pour la validation, avec des annotations pour la classe 'pothole'. Ce dataset est diversifié (conditions d'éclairage et météo variées), ce qui est un bon point de départ pour adapter aux routes camerounaises et est accessible à l'adresse : https://www.kaggle.com/datasets/anggadwisunarto/potholes-detection-yolov8.
2. **Téléchargement :** Nous avons créé un compte Kaggle, téléchargé le ZIP (environ 755 MB), et décompressé le dossier. Structure obtenue :

   — train/images/ et train/labels/ (images JPG + annotations TXT)
   — valid/images/ et valid/labels/
   — data.yaml
   — Une vidéo sample (sample_video.mp4) pour tests.

III Création de Compte Roboflow et Import du Dataset Public

— **Pourquoi Roboflow ?** : Pour faciliter l'upload, l'annotation, les augmentations et l'export au format YOLOv8. Nous avons créé un compte gratuit sur Roboflow.com, puis un projet "Object Detection" nommé "Pothole Cameroon".

— **Import du Dataset Kaggle** : Nous avons uploadé le ZIP entier (train + valid + data.yaml). Roboflow a détecté le format YOLO et importé les 1977 images annotées.

— Après l'annotation, nous avons obtenu 1547 images redimensionnées en 512 × 512, puis nous avons créé une nouvelle version du dataset, que nous avons appelée : **images_ok** pour respecter la consigne 70-15-15 et que nous avons importée au format **YOLOv8**. Cette nouvelle version est accessible à l'adresse : https://app.roboflow.com/pothole-detection-wipyl/pothole-detection-dschang-bafous/1

IV Préparation de l'environnement pour l'Entraînement

— **Outils :** Google Colab pour GPU, Google drive pour importer le dataset

V Entraînement du Modèle

Le notebook est accessible à l'adresse : https://colab.research.google.com/drive/1ABJ9w4BZfpVnUM0rG-9DTV8EWPE-QkoT?usp=drive_link

VI Évaluation du Modèle

Après l'entraînement sur 100 epochs, le modèle a été validé automatiquement sur l'ensemble de validation. Voici la sortie complète générée par Ultralytics (YOLOv8) et son interprétation détaillée.

```
100 epochs completed in 0.678 hours.
Optimizer stripped from /content/runs/detect/train/weights/last.pt, 6.3MB
Optimizer stripped from /content/runs/detect/train/weights/best.pt, 6.3MB

Validating /content/runs/detect/train/weights/best.pt...
Ultralytics 8.4.12 🚀 Python-3.12.12 torch-2.9.0+cu126 CUDA:0 (Tesla T4, 15095MiB)
Model summary (fused): 73 layers, 3,005,843 parameters, 0 gradients, 8.1 GFLOPs
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% ━━━━━━━━━━━━ 8/8 2.2it/s 3.7s
                   all        232        884      0.768      0.683      0.758      0.488
Speed: 0.2ms preprocess, 2.3ms inference, 0.0ms loss, 3.5ms postprocess per image
Results saved to /content/runs/detect/train
```

FIGURE 4 – Evaluations du modèle

— **100 epochs completed in 0.678 hours**
L'entraînement a duré environ 40 minutes (0.678 heures) pour 100 epochs. Cela montre une vitesse très correcte grâce à l'utilisation du GPU Tesla T4 sur Google Colab.

— **Model summary (fused) : 73 layers, 3,005,843 parameters, 8.1 GFLOPs**

  — **73 layers** : le modèle YOLOv8n (nano) fusionné (optimisé pour l'inférence) contient 73 couches au total.

  — **3 millions de paramètres** : très léger → rapide et adapté aux déploiements embarqués.

  — **8.1 GFLOPs** : mesure de la complexité computationnelle → faible, donc très rapide à l'inférence.

— **Class Images Instances Box(P R mAP50 mAP50-95) : 100% 8/8 00 :03 2.37it/s**

  — **all** : résultats globaux sur toutes les classes (ici seulement 'pothole').

  — **Images : 232** : nombre d'images dans l'ensemble de validation.

  — **Instances : 884** : nombre total d'objets (nids-de-poule) annotés dans ces images.

  — **Box(P)** : Précision (Precision) = 0.768 → 76,8% des détections sont correctes (faible taux de faux positifs).

  — **Box(R)** : Rappel (Recall) = 0.683 → 68,3% des vrais nids-de-poule ont été détectés.

- **mAP50** : mean Average Precision à IoU=0.5 = 0.758 → **75,8%** → très bon résultat (objectif > 0.70 atteint).
- **mAP50-95** : mAP moyen sur IoU de 0.5 à 0.95 = 0.488 → 48,8% → montre que pour des seuils IoU plus stricts (boîtes très précises), la performance diminue (classique pour YOLO sur objets irréguliers comme les potholes).

— **Speed : 0.2ms preprocess, 2.3ms inference, 3.5ms postprocess per image**

- **2.3 ms d'inférence par image** → environ **435 images par seconde** sur Tesla T4 (très rapide).
- Temps total par image = 6 ms

— **Results saved to /content/runs/detect/train**
Tous les résultats (courbes de perte, matrice de confusion, images de validation annotées, best.pt, last.pt) sont sauvegardés dans ce dossier.

VI.1 Interprétation globale des performances

— **mAP@0.5 = 75,8%** → Excellent pour un premier entraînement sur un dataset mixte (Kaggle + local). Le modèle détecte bien les nids-de-poule dans la majorité des cas.

— **Precision (76,8%) > Recall (68,3%)** → Le modèle fait peu de fausses détections (bon pour éviter les alertes inutiles), mais manque encore quelques petits ou mal éclairés potholes.

— **mAP50-95 = 48,8%** → Indique que les boîtes prédites ne sont pas toujours très précises (forme irrégulière des potholes). Améliorable avec plus d'images locales et augmentations.

— **Vitesse** : largement au-dessus de l'objectif (> 20 FPS).

VII Déploiement du Modèle

Nous avons créé une application flask pour déployer le modèle obtenu.

Le dossier complet du projet est accessible à l'adresse : https://github.com/VirlenceDongmo/Pothole_Detection.git

Bibliographie

[1] Joseph Redmon et al. « You only look once : Unified, real-time object detection ». In : *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, p. 779-788.