Résumé sur YOLOv8 et ses Composants

1 Structure générale de YOLOv8

YOLOv8, comme ses prédécesseurs, se compose principalement de deux parties : le **Backbone** et le **Head**.

- Le *Backbone* est responsable de l'extraction des caractéristiques de l'image d'entrée. Il utilise une série de convolutions et autres opérations pour réduire progressivement la dimension spatiale de l'image tout en augmentant le nombre de canaux de caractéristiques.
- Le *Head* sert à la détection d'objets proprement dite, prenant les caractéristiques extraites par le Backbone et les utilisant pour prédire les classes d'objets, les boîtes englobantes et les scores de confiance.

2 Le modèle de détection YOLOv8

Cette section décrit en détail la structure du Backbone et du Head du modèle Yolov8 de détection d'objet. Le nombre de caneaux et la profondeur du modèle sont dynamiques.

2.1 Backbone

Le Backbone de YOLOv8 est conçu pour extraire efficacement des caractéristiques de différentes échelles à partir de l'image d'entrée. Il est composé d'une série de couches de convolution et de blocs C2f, chacun jouant un rôle clé dans le traitement des informations visuelles.

- Les couches de **convolution initiales** servent à réduire la dimension spatiale de l'image tout en augmentant progressivement le nombre de

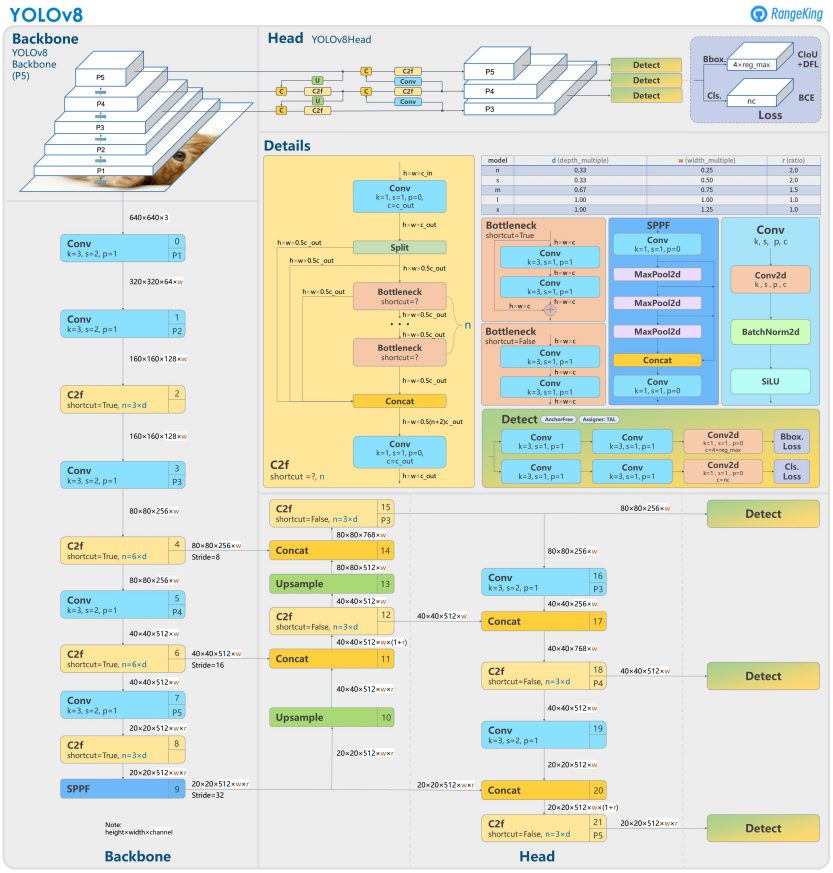

Figure 1: Architecture du modèle de détection Yolov8

canaux. Par exemple, la première couche applique une convolution avec 64 filtres de taille 3×3 avec un pas de 2, réduisant de moitié la résolution spatiale (*P1/2*).

- Les **blocs C2f** sont répétés plusieurs fois à différentes profondeurs et avec différents nombres de canaux, permettant au modèle de capturer des caractéristiques complexes à différents niveaux de granularité.

- Le Backbone intègre également un **bloc SPPF** après la dernière couche de convolution, utilisant le pooling spatial pyramidal pour agréger les informations contextuelles sur différentes régions de l'image, améliorant ainsi la robustesse du modèle aux variations de taille des objets.

2.2 Head

Le Head du modèle YOLOv8 est responsable de la prédiction des boîtes englobantes, des classes d'objets et des scores de confiance, en utilisant les caractéristiques extraites par le Backbone.

- Les couches **Upsample** et **Concat** permettent d'agréger les caractéristiques à différentes échelles, facilitant la détection d'objets de tailles variées. Ces opérations fusionnent les informations provenant de différentes parties du Backbone, créant une représentation enrichie des données. Voir les modèles de type FPN (Feature Pyramid Network) et PAN (Path Aggregation Network) [4]
- On y retrouve à nouveau des **blocs C2f**.
- Enfin, la couche **Detect** fait les prédictions finales. Elle utilise les caractéristiques traitées pour prédire les boîtes englobantes, les classes et les scores de confiance pour chaque objet détecté dans l'image, en prenant en compte les caractéristiques de différents niveaux de profondeur (*P3*, *P4*, *P5*).

3 Yolov8 pour de la classification binaire

Un modèle de Yolov8 utilisé pour de la classification est présenté dans le code source d'ultralytics. Il s'agit d'un modèle dont le backbone est identique au modèle de détection excepté le dernier bloc SPPF. Pour le head, un simple bloc de classification est utilisé. On pourra ajouter une fonction d'activation sigmoïde en dernière couche afin de normaliser les sorties en fonction de probabilité d'appartenance à une classe.

4 Description des blocs

4.1 Le bloc Conv

Le bloc Conv est constitué de trois couches :

1. Une couche de convolution : Conv2d
2. Une couche de normalisation par batch (BatchNorm2d)

3. Et enfin une couche d'activation SiLu (pour Sigmoid Linear Unit)

4.2 Le bloc C2f

Le bloc `C2f` est une implémentation optimisée qui fait partie de l'architecture plus large des réseaux de neurones convolutifs, notamment utilisée dans les modèles de type YOLOv8. Ce bloc est une variation du concept de Bottleneck Cross Stage Partial (CSP) [2], conçu pour maximiser l'efficacité computationnelle tout en conservant une capacité significative d'extraction de caractéristiques. Le bloc `C2f` est structuré comme suit :

1. **Réduction des canaux d'entrée** : La première couche de convolution dans le bloc `C2f` a pour rôle de réduire le nombre de canaux d'entrée. Cette étape utilise des filtres de convolution 1x1 pour comprimer l'information, réduisant ainsi la dimensionnalité des caractéristiques. Cette réduction préalable aide à alléger la charge computationnelle des étapes suivantes.

2. **Traitement par blocs Bottleneck** : Après la réduction des canaux, les données passent à travers une série de blocs *Bottleneck*. Chaque bloc *Bottleneck* est conçu pour traiter efficacement les caractéristiques tout en conservant les informations essentielles. Dans ce contexte, les blocs *Bottleneck* utilisent une combinaison de convolutions 1x1 et 3x3. La convolution 1x1 sert à modifier le nombre de canaux tout en réduisant le coût computationnel, tandis que la convolution 3x3 traite spatialement les caractéristiques. Les blocs *Bottleneck* peuvent également incorporer des raccourcis (shortcuts), facilitant la propagation du gradient et améliorant ainsi la capacité d'apprentissage du réseau sur des architectures profondes. Ce type de bloc n'est pas spécifique à Yolo mais il est généralement utilisé dans les derniers modèles de CNN (de type Resnet) permettant une meilleure propagation du gradient dans le réseaux.

3. **Restauration des canaux de sortie** : La dernière étape dans le bloc `C2f` utilise une autre couche de convolution pour ajuster le nombre de canaux de sortie à la dimensionnalité désirée.

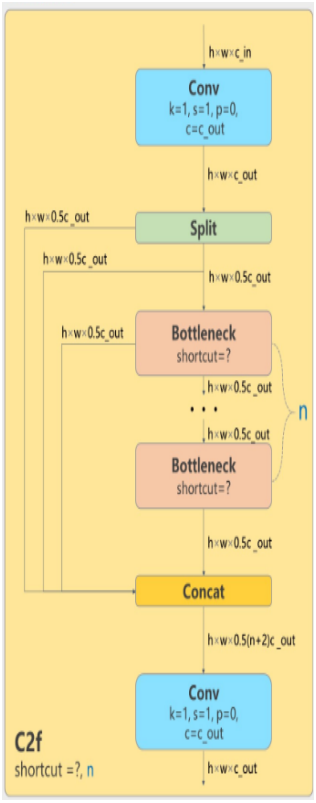

Figure 2: Architecture du bloc C2f

4.3 Le bloc SPPF [3]

Le bloc SPPF (Spatial Pyramid Pooling - Fast version) est une amélioration de l'opération de pooling spatial pyramidal traditionnelle, spécifiquement conçue pour augmenter l'efficacité computationnelle dans les architectures de réseaux neuronaux convolutifs. L'objectif principal du bloc SPPF est de capturer des caractéristiques contextuelles à différentes échelles.

Le bloc SPPF réalise un pooling pyramidal en utilisant plusieurs fenêtres de pooling de tailles différentes en parallèle, pour ensuite concaténer les sorties. C'est ce qui permet notamment de détecter des objets de différentes tailles dans une image.

4.4 Le bloc Detect

La classification et la détection des boîtes englobantes se font dans le bloc Detect. La sortie est un tenseur de $N = N_c + 4 * N_{scale}$ canaux par ancre où N_c est le nombre de classe et N_scale est un nombre de canaux (dépendant de la taille du modèle) qui enregistre les informations des boxes prédites (x,y,h,w). Il semblerait que les opérations qui conduisent aux Nc et N_{scale} se déroulent en parallèlent dans des bloc `Cv2` et `Cv3` qui sont un ensemble de couches convolutionnelles. Les sorties de ces couches sont ensuite concaténées puis traitées par un décodeur pour fournir les localisation des boxes englobantes sur l'image.

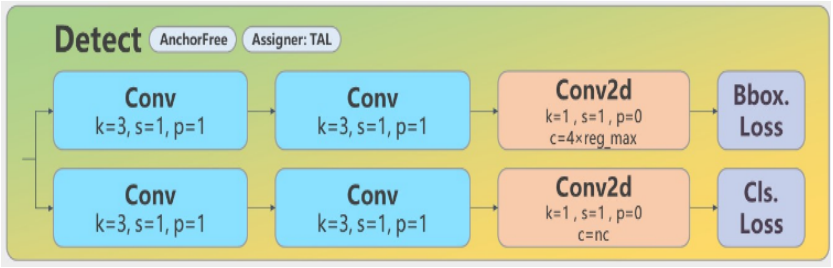

Figure 3: Architecture du bloc Detect

4.5 Le bloc Classify

Ce bloc est constitué des couches suivantes :

1. Bloc Convolution
2. Couche de Pooling Adaptatif : Réalise une opération de pooling avec le pas et la taille du kernel qui s'adapte automatiquement en fonction de la dimension de sortie désirée.
3. Couche de Dropout : Désactive aléatoirement une certaine proportion des neurones durant l'apprentissage forçant le modèle à apprendre de façon plus robuste.
4. Couche Linéaire : Apllique une transformation linéaire aux données d'entrées afin de les transformer en scores de classe. Il y a autant de sorties qu'il y a de classes.

Pour une classification binaire, il est courant d'utiliser une fonction d'activation sigmoid sur la drnière couche pour convertir le score linéaire en une probabilité comprise entre 0 et 1.

5 Les métriques de détection d'objet

5.1 Le Best Possible Recall (BPR)

Le BPR, ou "Best Possible Recall" (meilleur rappel possible), est une métrique utilisée dans le domaine de la détection d'objets pour évaluer la capacité d'un modèle à identifier correctement les objets d'intérêt. Le rappel, dans ce contexte, est la proportion d'objets réels correctement détectés par le modèle par rapport au nombre total d'objets réels présents. Ainsi, le BPR mesure la capacité maximale du modèle à récupérer ces objets sans tenir compte du nombre de fausses détections qu'il pourrait générer.

Le BPR est calculé en ajustant le seuil de détection du modèle. Pour chaque objet dans une image, un modèle de détection d'objets produit un score de confiance indiquant la probabilité que l'objet détecté appartienne à une certaine classe. En ajustant le seuil de score de confiance pour la détection (par exemple, en considérant comme détecté tout objet avec un score supérieur à ce seuil), on peut influencer le nombre d'objets détectés par le modèle. Le BPR est obtenu en réduisant ce seuil au point où tous les objets réels sont détectés (maximisant le rappel), indépendamment du nombre de fausses positives (détections incorrectes) générées.

Le BPR est particulièrement utile pour comprendre la limite supérieure de la performance d'un modèle de détection d'objets.

5.2 La métrique d'alignement

La métrique d'alignement permet d'évaluer à la fois la tâche de classification et la tâche de détection. C'est une métrique assez simple dépendant de deux paramètres α et β qui pondèrent respectivement les scores de classification s_{class} et de détection s_{detect}.
En pratique le score de détection utilisé est tout simplement l'IoU (Intersection over Union). La formulation de cette metrique est la suivante :

$$M_{align} = s_{class}^{\alpha} s_{detect}^{\beta} \tag{1}$$

Le score de classification quand a lui est plutôt un score de confiance ou probabilité d'appartenance d'un objet à cet ancre.

6 Les fonctions de coûts en détection d'objet

6.1 La notion d'ancrage

Afin de comprendre comment son établies les fonction de coût liées à la détection d'objet, il est essentiel de tout d'abord étudier la notion d'ancres (Anchor). En effet les fonctions de coûts évaluent la prédiction du modèle localement sur chaque niveau de caractéristiques.
Pour ce faire, considérons un tenseur représentant l'espace de caractéristiques extraites sur un niveau quelconque du modèle. Par exemple au niveau 0, l'espace de caractéristique est la donnée d'entrée donc un tenseur de dimension 640*640*3 si c'est une image rgb de taille 640*640. Sur chacun d'eux on établie une liste de points d'ancrage qui sont des points prédéfinis localisés sur l'espace des dimensions extraites et qui permettrons d'évaluer les résultats de prédiction localement (donnant de meilleures résultats qu'en les évaluant globablement).
Dans Yolov8, cette liste de points d'ancrage est définie très simplement :

1. On quadrille chaque niveau de caractéristiques avec un nombre de caractéristiques fixes dans la grille.

2. Les points d'ancrage sont les centres de chacunes de ces grilles.

Ainsi, on pourra évaluer les prédictions du modèles sur différentes échelles car la taille relative des grilles, comparée à la dimension de la donnée d'entrée, évolue lorsqu'on change de niveau dans les couches du modèle. Par exemple, en prenant un pas de 8 pour la quadrillage, une grille sur le premier tenseur (l'image d'entrée de dimension $640 \times 640 \times 3$) représente une taille relative de $12.5\% \times 12.5\%$ alors que sur un niveau différent où l'échelle de caractéristiques est $320 \times 320 \times 64$, la taille relative a doublé.

6.2 Les prédictions du modèle

Une fonction de coût évalue les prédictions du modèle par rapport aux valeurs réelles. Les fonctions de coût sont appliquées sur chaque ancre de l'espace

des caractéristiques extraites. Quelles sont les valeurs prédites par le modèle par ancre ?
Parmis les boîtes prédites, un nombre important d'entres elles ne sont pas utiles, ainsi on sélectionne un nombre limité classé suivant la métrique d'alignement définie en 5.2
Puis on assigne à chacune de ces boîtes prédites une ancre en fonction des coordonnées de son centre.
On aimerait obtenir qu'une seule prédiction par ancre, de ce fait, on assigne a l'ancre celle qui a la plus grande iou Ainsi pour les ancres sélectionnées, les prédictions contiennent les coordonnées de la boîte englobante, en générale (x_c,y_c,h,w) ou (x_{min},x_{max},y_{min},y_{max}) ou tout simplement (x_1,x_2,y_1,y_2) pour les boîtes orientées. Ils contiennent aussi le score de prédiction c (score de confiance du modèle sur le fait que l'ancre contienne un objet) et les scores de classes (s_{c1}, s_{c2}, ..., s_{cn})

6.3 Classification Loss

Dans la suite nous allons noter la fonction de coût liée à la classe c pour l'ensemble des vecteurs de prédiction du batch x et y ses vecteurs associés :

$$l_c(x, y) = l_{1,c}, l_{2,c}, ..., l_{N,c} \tag{2}$$

N est le nombre de vecteur dans le batch. Ainsi $l_{n,c}$ est le scalaire de coût du vecteur x_n associé la classe c.
La fonction de coût totale étant :

$$L(x, y) = \sum_c l_c(x, y) \tag{3}$$

La fonction de coût appliquée au batch est sa forme réduite soit en prenant la moyenne de (2) soit la somme.
On notera aussi p_c la probabilité de classification du vecteur x à la classe c.
Enfin pour simplifier les notations, nous prendrons une taille de batch $N = 1$.

6.3.1 Cross Entropy Loss (CEL)[5]

Il est important de tout d'abord introduire la fonction de coût classiquement appliquée dans les problèmes de classification en Deep Learning. La formule

de la Binary Cross Entropy (BCE) est la suivante :

$$L_{BCE}(p_c) = \begin{cases} -\log(p_c) & \text{si } y = 1 \\ -\log(1 - p_c) & \text{sinon} \end{cases} \tag{4}$$

Celle de la Categorical Cross Entropy (CCE) est :

$$L_{CSE} = -\sum y \log(p_c) \tag{5}$$

Où $y = +-1$ spécifie la classe concernée. La BCE est utilisée pour les problèmes de classification binaire, où chaque entrée est classée comme appartenant à une des deux catégories. La CSE est utilisée pour les problèmes de classification multiclasse, où chaque entrée peut appartenir à une parmi plusieurs catégories. Elle calcule une perte séparée pour chaque classe et les additionne.

6.3.2 Focal Loss [5]

Introduit en 2017 avec notamment l'architecture CNN RetinaNet, la Focal Loss est utilisée pour traiter la problématique de classification dans les modèles de détection d'objets.

$$l_{c,FL}(p_c) = \begin{cases} -\alpha(1 - p_c)^\gamma \log(p_c) & \text{si } y = 1 \\ -p_c(1 - \alpha)^\gamma \log(1 - p_c) & \text{sinon} \end{cases} \tag{6}$$

Où γ est un hyperparamètre qui réduit la contribution des exemples faciles et augmente celle des exemples difficiles, et α est un hyperparamètre qui équilibre la perte entre les classes positives et négatives.
L'avantage que présente la FL comparée à la CEL est qu'elle ne traite pas de la même façon les exemples simples des complexes et de même elle considère la proportions des classes dans le jeux de données.

6.3.3 Varyfocal Loss (VFL) [6]

Dans Yolov8, la fonction de coût utilisée pour la classification est une variation de Focal Loss apparu le modèle de détection d'objet VFNet en 2021. L'objectif étant d'y ajouter le paramètre de confiance du modèle dans sa

prédiction ainsi que de prendre en compte la prédiction de localisation des boîtes englobante.

$$l_{c,VFL}(p) = \begin{cases} -q_c(q_c \log(p_c) + (1 - \log(q_c)) \log(1 - p_c)) & \text{si } q_c > 0 \\ -\alpha p_c^{\gamma} \log(1 - p_c) & \text{si } q_c = 0 \end{cases} \tag{7}$$

Ici p_c est le score IACS (IoU Aware Classification Score) de la classe concernée. L'IACS est un vecteur qui vaut l'IoU (Intersection over Union) entre la boîte prédite et la boîte réelle sur l'indice de classe réelle et 0 pour tout autre indice. q_c est le score de confiance de prédiction donnée par le modèle pour cette classe (à vérifier si ce n'est pas le score de confiance globale). On remarque que contrairement à la FL, les prédictions positives et négatives ne sont pas traitées de façon symétriques. On accorde plus d'importances aux exemples positifs qui sont plus rares en pratique. Aussi on pondère par le facteur q, ainsi, les IoU élevés auront un poids plus important lors de l'apprentissage. α est toujours utilisé pour pondérer le poids entre les exemples positifs et négatifs.

6.3.4 Modified BCEL (MBCEL)

On peut voir qu'en pratique, le modèle de détection Yolov8 n'utilise pas la fonctionVFL mais plutôt une fonction modifiée de la fonction de coût BCE. Cette fonction est disponible dans pythorch sous le nom de BCEWITHLOGITSLOSS (torch.nn.BCEWITHLOGITSLOSS). Sa formule est la suivante :

$$l_{c,MBCEL}(p_c) = \begin{cases} -w_c \alpha_c y \log(p_c) & \text{si } y = 1 \\ -w_c(1 - y) \log(1 - p_c) & \text{si } y = 0 \end{cases} \tag{8}$$

Où α est le facteur de proportion de la classe c dans l'ensemble de données et w_c est un facteur de poids.

6.4 Detection Loss

Dans la suite nous noterons $B = b_1, ..., b_m$ les boites englobantes prédites et $G = g_1, ..., g_n$ les boîtes englobantes réelles.

6.4.1 IoULoss

Le principe de cette fonction de coût est d'évaluer la détection en termes de proportion de superposition par rapport aux boites réelles. Sa formule est la

suivante :

$$L_{IoU} = 1 - IoU(b_i, g_j) \tag{9}$$

6.4.2 GIoULoss [7]

"Generalized Intersection over Union Loss" est la première metrique introduite dans l'apprentissage des algorithmes de détection d'objet qui considère l'IoU entre les boîtes englobantes prédites et celles réelles. L'IoU seule ne permet pas de fournir un gradient utile lors de l'apprentissage car elle ne permet pas de distinguer entre deux objets ne se chevauchant pas mais l'un étant plus éloigné que l'autre. L'idée est donc de considérer C la plus petite boîte englobant la prédiction et la vérité.

$$L_{GIoU} = 1 - GoU(b_i, g_j) = 1 - IoU(b_i, g_j) - \frac{|C \backslash (b_i \cup g_i)|}{|C|} \tag{10}$$

6.4.3 DIoULoss [8]

Afin d'améliorer la vitesse de convergence du gradient lors de l'apprentissage, Distance IoULoss a été introduit et considère la distance entre les centres des boites prédites et réelles. En notant $\rho(A_c, B_c)$ la distance euclidenne entre A_c et B_c, les centres des boites A et B :

$$L_{DIoU} = 1 - IoU(b_i, g_j) + \frac{\rho^2(b_{ci}, g_{ci})}{d_C^2} \tag{11}$$

Ici d_C est la diagonale de la plus petite boîte englobante qui couvre à la fois les boîtes prédites et réelles, servant à normaliser la distance entre les centres.

6.4.4 CIoULoss [8]

CIoULoss est une version améliorer de DIoULoss dont l'idée est de considérer le rapport de forme ν entre la boîte prédite et la boîte réelle. En pratique, ν se calcule de la façon suivante :

$$\nu = \frac{4}{\pi^2}(\arctan \frac{w_{g_j}}{h_{g_j}} - \arctan \frac{w_{b_i}}{h_{b_i}})^2$$

Où w et h indiquent respectivement la longueur et la hauteur de la boîte. Ainsi :

$$L_{CIoU} = 1 - IoU(b_i, g_j) + \frac{\rho^2(b_{ci}, g_{ci})}{d_C^2} - \alpha\nu \tag{12}$$

Où α est un facteur de pondération.

6.4.5 DFLoss [9]

Distribution Focal Loss est une manière de considérer la tâche de détection de boîte englobante comme une regression plutôt qu'une classification. L'idée est de discrétiser les valeurs pouvant être prises par chaque coordonnée de boîte englobante correspondant à la sortie d'une couche softmax du réseau. On les notes : $[y_0, ..., y_n]$ avec n qui définit le pas de discrétisation. L'objectif étant que la densité de probabilité $P(.)$ retournée par la couche softmax soit maximum au niveau de la valeur réelle y, c'est à dire aux alentours de y_i et y_{i+1}, on y applique une cross entropy pour obtenir le scalaire de perte :

$$L_{DFL}(S_i, S_{i+1}) = -\left[(y_{i+1} - y)\log(S_i) + (y - y_i)\log(S_{i+1})\right] \tag{13}$$

Où S_i et S_{i+1} sont respectivement $P(y_i)$ et $P(y_{i+1})$ la densité de probabilité retournée par la couche softmax à l'indice i et $i + 1$.

6.4.6 BboxLoss

7 Yolov8 d'ultralytics

Ultralytics est la structure fournissant un code d'ensemble pour Yolov8 disponible en open-source sur Github. C'est ce code source qui est utilisé dans notre étude, les parties importantes sont décrites ci-dessous.

- Un ficher YAML permettant de gérer dynamiquement la construction de son modèle
- Une fonction `parse_model` qui construit le modèle en pytorch à partir du fichier YAML

7.1 Description du fichier YAML

Le fichier YAML pour YOLOv8 contient la configuration du modèle, incluant le nombre de classes, l'activation, les échelles de modèle, et les spécifications des couches pour le Backbone et le Head. Les différents champs du fichier YAML sont :

- Le champ *nc* définit le nombre de classes pour la tâche de détection.
- *Activation* précise la fonction d'activation à utiliser dans le réseau.
- Les *scales* permettent d'ajuster la complexité du modèle en modifiant la profondeur et la largeur des couches.
- Les sections *backbone* et *head* décrivent les couches spécifiques et leurs configurations.

Un ensemble de fichier YAML avec des modèles prédéfinis sont disponibles ici : ultralytics/cfg/models/v8/

7.2 Utilisation de la fonction `parse_model`

La fonction `parse_model` prend en entrée le dictionnaire YAML et le nombre de canaux d'entrée, et construit le modèle PyTorch correspondant. Elle gère dynamiquement les dimensions des canaux à travers le réseau et applique les modifications spécifiées par les échelles de modèle. Elle est disponible ici : ultralytics/nn/tasks.py

7.3 Les fonction de coûts

Dans le module d'ultralytics, on retrouve les différentes fonctions de coûts évoqués en partie 6. ici : ultralytics/utils/loss.py En particulier les classes suivantes :

1. v8DetectionLoss qui calcule le coût pour un batch de données prédit pour le modèle de détection Yolov8
2. v8ClassificationLoss qui calcul le coût pour un batch de données prédit pour un modèle de classification utilisant le backbone de Yolov8

7.4 Les fonctions utiles

On peut y retrouver la fonction qui réalise les ancres (la grille d'ancre sur l'image ou le batch d'image) : ultralytics/utils/tal.py => make anchors
L'ensemble des metrics utilisées : ultralytics/utils/metrics.py
La fonction qui attribue les prédictions aux ancres sur chaque couche du modèle : ultralytics/utils/tal.py => TaskAlignedAssigner

8 Glossaire

PyTorch : Un framework open source de machine learning notamment en python.

Modèle (de type modèle de deep learning) : Une architecture de réseau de neurones.

Fonction d'activation : Une fonction mathématique appliquée à l'entrée d'un neurone dans un réseau neuronal, déterminant l'activation du neurone. Les fonctions d'activation les plus courantes incluent ReLU, Sigmoid et Tanh.

Opération et couche de pooling : Une technique utilisée pour réduire la dimensionnalité spatiale des représentations d'entrée, en conservant les informations les plus importantes.

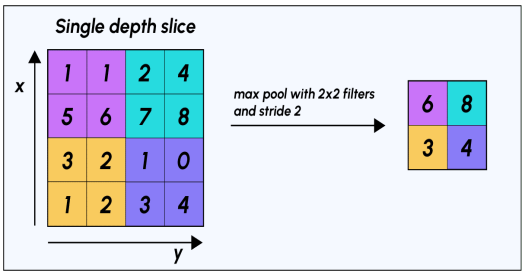

Figure 4: Max Pooling

Canaux : Dans le traitement d'images et les réseaux de neurones, les canaux font référence aux dimensions distinctes des données d'entrée. Dans les réseaux neuronaux convolutifs, chaque canal peut également représenter une caractéristique ou un motif appris à partir des données d'entrée.

Granularité : Le niveau de détail ou de spécificité des caractéristiques que le modèle peut apprendre et reconnaître dans les données.

Ancres/Anchor : Dans la détection d'objets, les ancres sont des boîtes prédéfinies de différentes échelles et ratios d'aspect qui servent de points de référence pour prédire les boîtes englobantes des objets détectés.

Couches sigmoïde : Une couche ou une fonction d'activation qui produit une sortie comprise entre 0 et 1, souvent utilisée pour prédire la probabilité dans les tâches de classification binaire.

Boxes englobantes : Dans la tâche de détection d'objets, les boxes englobantes sont des rectangles qui entourent les objets détectés dans une image, indiquant la position et la taille de chaque objet.

References

[1] Juan Terven, Diana-Margarita, Córdova-Esparza and Julio-Alejandro, Romero-González , A Comprehensive Review of YOLO Architectures in Computer Vision: From YOLOv1 to YOLOv8 and YOLO-NAS

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, , Deep Residual Learning for Image Recognition

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun , Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition

[4] Shu Liu, Lu Qi, Haifang Qin, Jianping Shi, Jiaya Jia , Path Aggregation Network for Instance Segmentation

[5] Tsung-Yi Lin, Priya Goyal, Ross, Girshick, Kaiming He, Piotr Dollar , Focal Loss for Dense Object Detection 2017

[6] Haoyang Zhang, Ying Wang, Feras Dayoub, Niko Sunderhauf , VarifocalNet: An IoU-aware Dense Object Detector

[7] Hamid Rezatofighi,Nathan Tsoi, JunYoung Gwak, Amir Sadeghian, Ian Reid, Silvio Savarese , Generalized Intersection over Union: A Metric and A Loss for Bounding Box Regression, 2019

[8] Zhaohui Zheng, Ping Wang, Wei Liu, Jinze Li, Rongguang Ye, Dongwei Ren , Distance-IoU Loss: Faster and Better Learning for Bounding Box Regression

[9] Xiang Li, Wenhai Wang, Lijun Wu, Shuo Chen, Xiaolin Hu, Jun Li, Jinhui Tang and Jian Yang , Generalized Focal Loss: Learning Qualified and Distributed Bounding Boxes for Dense Object Detection