

Cours Machine Learning

Chapitre 13

Vision par Ordinateur Avancée

Objectifs d'apprentissage :

- Maîtriser les architectures de détection d'objets (R-CNN, YOLO, Faster R-CNN)
- Comprendre la segmentation sémantique et d'instances (U-Net, Mask R-CNN)
- Découvrir les Vision Transformers (ViT) et leur application
- Comprendre les modèles vision-langage (CLIP)
- Implémenter des pipelines de détection et segmentation

Prérequis : Chapitres 06 (MLP), 07 (CNN), 08 (Transformers)

Durée estimée : 8-10 heures

Notebooks : 12*demo*.ipynb*

Table des matières

1	Motivation	2
2	Détection d'Objets (Object Detection)	2
2.1	Définition du Problème	2
2.2	Métriques d'Évaluation	2
2.2.1	Intersection over Union (IoU)	2
2.2.2	Mean Average Precision (mAP)	3
2.3	R-CNN (2014) - Région-based CNN	3
2.3.1	Pipeline R-CNN	3
2.3.2	Limites de R-CNN	3
2.4	Fast R-CNN (2015)	3
2.4.1	Architecture Fast R-CNN	4
2.5	Faster R-CNN (2015)	4
2.5.1	Architecture Faster R-CNN	4
2.5.2	Loss Function	5
2.6	YOLO (You Only Look Once)	5
2.6.1	Architecture YOLO (v1, 2016)	5
2.6.2	Loss Function YOLO	5
2.6.3	YOLOv2 et YOLOv3 (2017-2018)	6
2.6.4	YOLOv5, YOLOv8 (2020-2023)	6
2.7	Comparaison R-CNN vs YOLO	7
2.8	Non-Maximum Suppression (NMS)	7
3	Segmentation Sémantique	8
3.1	Définition du Problème	8
3.2	Métriques d'Évaluation	8
3.2.1	Intersection over Union (IoU) par classe	8
3.2.2	Mean IoU (mIoU)	8
3.2.3	Dice Coefficient	8
3.3	Fully Convolutional Networks (FCN, 2015)	8
3.3.1	Architecture FCN	9
3.4	U-Net (2015)	9
3.4.1	Architecture Détaillée	9
3.4.2	Loss Function U-Net	10
3.5	DeepLab (v1-v3, 2015-2018)	10
3.5.1	Atrous Spatial Pyramid Pooling (ASPP)	10
3.5.2	DeepLab v3+	11
3.6	Mask R-CNN (2017)	11
3.6.1	RoI Align	11
3.6.2	Loss Function	11
3.7	Comparaison des Architectures de Segmentation	12
4	Vision Transformers (ViT)	12
4.1	Motivation	12

4.2	Architecture Vision Transformer (ViT, 2020)	12
4.2.1	Patchification	12
4.2.2	Architecture Complète	13
4.2.3	Positional Encoding	13
4.3	Variantes de ViT	13
4.3.1	ViT-Base, ViT-Large, ViT-Huge	13
4.3.2	DeiT (Data-efficient image Transformers)	13
4.3.3	Swin Transformer (2021)	13
4.4	CNN vs ViT	14
5	Modèles Vision-Langage : CLIP	14
5.1	Motivation	14
5.2	Architecture CLIP	15
5.2.1	Composants	15
5.2.2	Loss Contrastive	15
5.3	Zero-Shot Classification	15
5.4	Résultats CLIP	15
6	Implémentation	16
6.1	Détection d'Objets avec YOLOv8	16
6.2	Segmentation avec U-Net (PyTorch)	16
6.3	Vision Transformer avec timm	18
6.4	CLIP Zero-Shot	19
7	Avantages et Limites	20
7.1	Object Detection	20
7.1.1	Avantages	20
7.1.2	Limites	20
7.2	Segmentation	20
7.2.1	Avantages	20
7.2.2	Limites	20
7.3	Vision Transformers	20
7.3.1	Avantages	20
7.3.2	Limites	21
7.4	CLIP	21
7.4.1	Avantages	21
7.4.2	Limites	21
8	Hyperparamètres et Tuning	21
8.1	Détection d'Objets	21
8.2	Segmentation	22
8.3	Vision Transformers	22
9	Applications Pratiques	22
9.1	Conduite Autonome	22
9.2	Imagerie Médicale	22

9.3	Reconnaissance Faciale	23
9.4	Commerce Électronique	23
9.5	Surveillance et Sécurité	23
10	Résumé du Chapitre	23
10.1	Points Clés	23
10.2	Formules Essentielles	24
10.3	Tableau Récapitulatif	24
11	Exercices	24
11.1	Questions de Compréhension	24
11.2	Exercices Pratiques	25
12	Pour Aller Plus Loin	25
12.1	Lectures Recommandées	25
12.1.1	Papers Fondateurs	25
12.1.2	Papers Récents	26
12.2	Ressources en Ligne	26
12.3	Datasets	26
12.4	Outils et Bibliothèques	26
12.5	Prochaines Étapes	26

1 Motivation

La vision par ordinateur classique (chapitre 07) nous a permis de classifier des images avec des CNN. Cependant, de nombreuses applications réelles nécessitent bien plus qu'une simple classification :

Exemple : Applications nécessitant la vision avancée

- **Conduite autonome** : Détecter et localiser les voitures, piétons, panneaux
- **Imagerie médicale** : Segmenter précisément les tumeurs, organes
- **Surveillance** : Suivre et identifier les personnes dans une vidéo
- **Commerce électronique** : Rechercher des produits par image
- **Robotique** : Manipuler des objets détectés dans une scène

Ces problèmes requièrent trois capacités avancées :

1. **Object Detection** : Où sont les objets ? (bounding boxes)
2. **Semantic Segmentation** : Quel est le label de chaque pixel ?
3. **Instance Segmentation** : Identifier chaque instance individuelle d'un objet

Ce chapitre explore les architectures deep learning qui résolvent ces tâches complexes.

2 Détection d'Objets (Object Detection)

2.1 Définition du Problème

Définition : Object Detection

La détection d'objets consiste à localiser et classifier simultanément plusieurs objets dans une image. Pour chaque objet, on prédit :

- Une **bounding box** : (x, y, w, h) où (x, y) est le coin supérieur gauche, w la largeur, h la hauteur
- Une **classe** : probabilités $P(c_i|\text{box})$ pour chaque classe c_i
- Un **score de confiance** : probabilité qu'il y ait un objet dans la box

2.2 Métriques d'Évaluation

2.2.1 Intersection over Union (IoU)

L'IoU mesure le chevauchement entre la box prédite et la ground truth :

$$\text{IoU} = \frac{\text{Aire}(\text{Box}_{\text{pred}} \cap \text{Box}_{\text{gt}})}{\text{Aire}(\text{Box}_{\text{pred}} \cup \text{Box}_{\text{gt}})} \quad (1)$$

- $\text{IoU} = 1.0$: chevauchement parfait
- $\text{IoU} = 0.0$: aucun chevauchement
- $\text{IoU} \geq 0.5$: généralement considéré comme une détection correcte

2.2.2 Mean Average Precision (mAP)

La métrique standard pour évaluer les détecteurs :

1. Pour chaque classe c , calculer la courbe Precision-Recall
2. Calculer l'aire sous la courbe (Average Precision, AP)
3. Faire la moyenne sur toutes les classes : $\text{mAP} = \frac{1}{C} \sum_{c=1}^C AP_c$
 - **mAP@0.5** : seuil IoU = 0.5
 - **mAP@0.5 :0.95** : moyenne sur seuils IoU de 0.5 à 0.95 (par pas de 0.05)

2.3 R-CNN (2014) - Région-based CNN

Définition : R-CNN

R-CNN propose de combiner :

1. **Selective Search** : algorithme traditionnel proposant 2000 régions candidates
2. **CNN** : extraction de features pour chaque région
3. **SVM** : classification de chaque région
4. **Régression** : ajustement des bounding boxes

2.3.1 Pipeline R-CNN

Algorithm 1 R-CNN

Require : Image I

Ensure : Liste de détections (boxes, classes, scores)

- 1 : Générer 2000 région proposals avec Selective Search
 - 2 : **for** chaque région R **do**
 - 3 : Redimensionner R en 227×227
 - 4 : Extraire features : $f_R = \text{CNN}(R)$
 - 5 : Classifier : $c = \text{SVM}(f_R)$
 - 6 : Ajuster box : $b' = \text{Regressor}(f_R)$
 - 7 : **end for**
 - 8 : Appliquer Non-Maximum Suppression (NMS)
 - 9 : **return** Détections filtrées
-

2.3.2 Limites de R-CNN

- **Très lent** : 47 secondes par image (2000 forward passes CNN)
- **Entraînement en 3 étapes** : CNN, SVM, régression (séparément)
- **Stockage important** : features extraites pour toutes les régions

2.4 Fast R-CNN (2015)

Idée clé : Ne calculer les features CNN qu'une seule fois pour toute l'image.

Définition : RoI Pooling

Le **Region of Interest (RoI) Pooling** permet d'extraire des features de taille fixe depuis n'importe quelle région de la feature map :

1. Projeter la région proposal sur la feature map
2. Diviser la région en $H \times W$ sous-régions
3. Appliquer max pooling sur chaque sous-région

Résultat : un vecteur de features de taille fixe $H \times W \times C$ pour chaque RoI.

2.4.1 Architecture Fast R-CNN**Algorithm 2** Fast R-CNN

Require : Image I , régions proposals $\{R_i\}$

Ensure : Détections

- 1 : Calculer feature map : $F = \text{CNN}(I)$ (une seule fois!)
- 2 : **for** chaque région R_i **do**
- 3 : Extraire features : $f_i = \text{RoIPool}(F, R_i)$
- 4 : Prédire classe : $P(c|R_i) = \text{FC}(f_i)$
- 5 : Ajuster box : $\Delta b_i = \text{FC}(f_i)$
- 6 : **end for**
- 7 : Appliquer NMS
- 8 : **return** Détections

Améliorations :

- **25x plus rapide** que R-CNN (0.32s par image)
- **Entraînement end-to-end** : une seule loss combinée
- **Multi-task loss** : classification + régression de box

2.5 Faster R-CNN (2015)

Idée clé : Remplacer Selective Search par un réseau de neurones.

Définition : Region Proposal Network (RPN)

Le RPN est un petit réseau fully convolutional qui prédit des région proposals directement depuis la feature map :

1. Faire glisser une fenêtre 3×3 sur la feature map
2. Pour chaque position, prédire k anchors boxes (différentes tailles/ratios)
3. Pour chaque anchor : prédire score objectness + ajustement de box

2.5.1 Architecture Faster R-CNN

1. **Backbone CNN** : ResNet-50, VGG, etc. \rightarrow feature map
2. **RPN** : propose des régions candidates
3. **RoI Pooling** : extrait features pour chaque proposition

4. Têtes de classification/régression : prédictions finales

2.5.2 Loss Function

Loss multi-task combinant RPN et détection :

$$L = L_{\text{RPN}}(\{p_i\}, \{t_i\}) + L_{\text{det}}(\{p'_i\}, \{t'_i\}) \quad (2)$$

où :

- $L_{\text{RPN}} = L_{\text{cls}}(p_i, p_i^*) + \lambda L_{\text{reg}}(t_i, t_i^*)$: loss du RPN
- L_{det} : loss de détection (similaire)
- p_i : probabilité objectness, t_i : coordonnées box

Améliorations :

- **10x plus rapide** que Fast R-CNN (0.2s par image, 5 FPS)
- **Entièrement appris** : plus besoin de Selective Search
- **État de l'art** en précision (mAP 70% sur COCO)

2.6 YOLO (You Only Look Once)

Philosophie différente : Faster R-CNN fait deux passes (RPN puis détection). YOLO fait tout en une seule passe forward !

Définition : YOLO

YOLO divise l'image en une grille $S \times S$ et prédit directement, pour chaque cellule :

- B bounding boxes avec leurs coordonnées (x, y, w, h)
- Un score de confiance par box : $P(\text{object}) \times \text{IoU}$
- Des probabilités de classe : $P(c_i | \text{object})$

Le réseau produit un tenseur de taille $S \times S \times (B \cdot 5 + C)$.

2.6.1 Architecture YOLO (v1, 2016)

1. 24 couches convolutionnelles (inspirées de GoogLeNet)
2. 2 couches fully connected
3. Sortie : $7 \times 7 \times 30$ (pour $S = 7$, $B = 2$, $C = 20$)

2.6.2 Loss Function YOLO

Loss complexe combinant 3 termes :

$$L = \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{K}_{ij}^{\text{obj}} [(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2] \quad (3)$$

$$+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{K}_{ij}^{\text{obj}} [(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2] \quad (4)$$

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{K}_{ij}^{\text{obj}} (C_i - \hat{C}_i)^2 \quad (5)$$

$$+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{K}_{ij}^{\text{noobj}} (C_i - \hat{C}_i)^2 \quad (6)$$

$$+ \sum_{i=0}^{S^2} \mathbb{K}_i^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2 \quad (7)$$

où $\mathbb{K}_{ij}^{\text{obj}}$ indique si un objet est présent dans la cellule i , box j .

2.6.3 YOLOv2 et YOLOv3 (2017-2018)

YOLOv2 (YOLO9000) :

- Batch Normalization dans toutes les couches
- Anchor boxes (comme Faster R-CNN)
- Haute résolution : 416×416 au lieu de 448×448
- Multi-scale training
- Darknet-19 backbone (19 couches)

YOLOv3 :

- Darknet-53 backbone (53 couches + residual connections)
- Prédiction multi-échelles (3 échelles : 13×13 , 26×26 , 52×52)
- Meilleure détection des petits objets
- Logistic regression pour objectness

2.6.4 YOLOv5, YOLOv8 (2020-2023)

YOLOv5 (Ultralytics) :

- Implémentation PyTorch moderne
- CSPDarknet backbone
- Auto-anchor, auto-learning bounding box anchors
- Mosaic augmentation
- **Très rapide** : 140 FPS sur GPU

YOLOv8 (2023) :

- Architecture améliorée (anchor-free)
- Meilleure précision (mAP 53% sur COCO)
- API simplifiée : `from ultralytics import YOLO`
- Support natif de la segmentation d'instances

TABLE 1 – Comparaison des architectures de détection

Modèle	mAP (%)	FPS	Approche	Temps réel
R-CNN	66.0	0.02	Two-stage	
Fast R-CNN	70.0	3.1	Two-stage	
Faster R-CNN	73.2	5	Two-stage	
YOLOv1	63.4	45	One-stage	
YOLOv3	57.9	65	One-stage	
YOLOv5	50.7	140	One-stage	
YOLOv8	53.9	80	One-stage	

2.7 Comparaison R-CNN vs YOLO

Astuce

Quand utiliser quoi ?

- **Faster R-CNN** : Précision maximale, applications non temps réel (imagerie médicale)
- **YOLO** : Temps réel, vidéo, applications embarquées (conduite autonome, surveillance)

2.8 Non-Maximum Suppression (NMS)

Problème : Les détecteurs produisent souvent plusieurs boxes pour le même objet.

Algorithm 3 Non-Maximum Suppression

Require : Boxes $B = \{b_1, \dots, b_n\}$, scores $S = \{s_1, \dots, s_n\}$, seuil IoU τ

Ensure : Boxes filtrées D

```

1 :  $D \leftarrow \emptyset$ 
2 : while  $B \neq \emptyset$  do
3 :    $b^* \leftarrow \operatorname{argmax}_{b \in B} S(b)$  (box avec le score max)
4 :    $D \leftarrow D \cup \{b^*\}$ 
5 :    $B \leftarrow B \setminus \{b^*\}$ 
6 :   for chaque box  $b_i \in B$  do
7 :     if  $\operatorname{IoU}(b^*, b_i) > \tau$  then
8 :        $B \leftarrow B \setminus \{b_i\}$  (supprimer box chevauchante)
9 :     end if
10 :  end for
11 : end while
12 : return  $D$ 

```

Variantes :

- **Soft NMS** : Au lieu de supprimer, réduire le score proportionnellement à l'IoU
- **DIoU-NMS** : Utiliser la Distance-IoU au lieu de l'IoU standard

3 Segmentation Sémantique

3.1 Définition du Problème

Définition : Segmentation Sémantique

La segmentation sémantique consiste à assigner une étiquette de classe à chaque pixel de l'image. Pour une image $I \in \mathbb{R}^{H \times W \times 3}$, on prédit une carte de segmentation $S \in \{1, \dots, C\}^{H \times W}$ où S_{ij} est la classe du pixel (i, j) .

Différence avec la détection :

- Détection : boxes rectangulaires
- Segmentation sémantique : contours précis au pixel près
- Segmentation d'instances : distingue les instances individuelles d'une même classe

3.2 Métriques d'Évaluation

3.2.1 Intersection over Union (IoU) par classe

$$\text{IoU}_c = \frac{TP_c}{TP_c + FP_c + FN_c} \quad (8)$$

où TP_c = pixels correctement prédits de classe c , FP_c = pixels faussement prédits, FN_c = pixels manqués.

3.2.2 Mean IoU (mIoU)

Moyenne de l'IoU sur toutes les classes :

$$\text{mIoU} = \frac{1}{C} \sum_{c=1}^C \text{IoU}_c \quad (9)$$

3.2.3 Dice Coefficient

Particulièrement utilisé en imagerie médicale :

$$\text{Dice} = \frac{2 \cdot |A \cap B|}{|A| + |B|} = \frac{2 \cdot TP}{2 \cdot TP + FP + FN} \quad (10)$$

Le Dice est équivalent à la F1-score pour la segmentation.

3.3 Fully Convolutional Networks (FCN, 2015)

Idée clé : Remplacer les couches fully connected par des convolutions pour produire une carte de segmentation.

Définition : FCN

Un FCN transforme un réseau de classification (VGG, ResNet) en réseau de segmentation :

1. Encoder : extraire features avec convolutions + pooling
2. Decoder : upsampling progressif pour retrouver la résolution originale
3. Skip connections : combiner features haute et basse résolution

3.3.1 Architecture FCN

1. **Convolutionalization** : Remplacer FC layers par convolutions 1×1
2. **Upsampling** : Transposed convolutions (deconvolutions) pour augmenter la résolution
3. **Skip connections** : Additionner les features du decoder avec celles de l'encoder

Variantes :

- **FCN-32s** : upsampling x32 en une seule étape
- **FCN-16s** : skip connection de pool4
- **FCN-8s** : skip connections de pool3 et pool4 (meilleur)

3.4 U-Net (2015)

Architecture emblématique pour la segmentation médicale.

Définition : U-Net

U-Net a une architecture en U symétrique :

- **Contracting path** (encoder) : Convolutions + max pooling ↓
- **Expansive path** (decoder) : Transposed convolutions ↑
- **Skip connections** : Concaténation (pas addition) des features

3.4.1 Architecture Détaillée

Encoder (Contracting Path):

Input (572x572x1)
 Conv 3x3 ReLU (570x570x64)
 Conv 3x3 ReLU (568x568x64)
 MaxPool 2x2 (284x284x64)
 Conv 3x3 ReLU (282x282x128)
 Conv 3x3 ReLU (280x280x128)
 MaxPool 2x2 (140x140x128)
 ... (4 niveaux au total)

Bottleneck:

Conv 3x3 ReLU (28x28x1024)
 Conv 3x3 ReLU (28x28x1024)

Decoder (Expansive Path):

UpConv 2x2 (56x56x512)

Concatenate avec skip connection de l'encoder
 Conv 3x3 ReLU
 Conv 3x3 ReLU
 ... (4 niveaux au total)

Output :

Conv 1x1 (388x388xC) pour C classes

Points clés :

- **Skip connections par concaténation** : préserve mieux les détails que l'addition
- **Data augmentation intensive** : rotations, déformations élastiques
- **Weighted loss** : pondérer la loss aux frontières entre cellules

3.4.2 Loss Function U-Net

Weighted cross-entropy pour gérer le déséquilibre de classes :

$$L = \sum_{x \in \Omega} w(x) \log(p_{l(x)}(x)) \quad (11)$$

où :

- $w(x)$: poids du pixel x (plus élevé aux frontières)
- $p_{l(x)}(x)$: probabilité softmax de la classe $l(x)$ au pixel x

Poids calculé pour séparer les instances proches :

$$w(x) = w_c(x) + w_0 \cdot \exp\left(-\frac{(d_1(x) + d_2(x))^2}{2\sigma^2}\right) \quad (12)$$

où $d_1(x)$, $d_2(x)$ sont les distances aux deux instances les plus proches.

3.5 DeepLab (v1-v3, 2015-2018)

Idée clé : Utiliser des **atrous convolutions** (dilated convolutions) pour augmenter le champ réceptif sans réduire la résolution.

Définition : Atrous Convolution

Une convolution atrous avec taux de dilatation r insère $r - 1$ zéros entre chaque poids du filtre :

$$y[i] = \sum_k x[i + r \cdot k] \cdot w[k] \quad (13)$$

Cela permet d'augmenter le champ réceptif de manière exponentielle sans augmenter le nombre de paramètres.

3.5.1 Atrous Spatial Pyramid Pooling (ASPP)

Module clé de DeepLab v2/v3 :

1. Appliquer des convolutions atrous avec différents taux : $r = \{6, 12, 18, 24\}$

2. Appliquer global average pooling
3. Concaténer toutes les features
4. Convolution 1×1 pour fusion

Cela capture le contexte à plusieurs échelles.

3.5.2 DeepLab v3+

Amélioration avec un decoder :

- Encoder : ResNet + ASPP
- Decoder : Upsampling progressif avec skip connections
- État de l'art sur PASCAL VOC (mIoU 89%)

3.6 Mask R-CNN (2017)

Extension de Faster R-CNN pour la segmentation d'instances.

Définition : Mask R-CNN

Mask R-CNN ajoute une branche de segmentation parallèle à Faster R-CNN :

1. Backbone + RPN + RoI Align (amélioration de RoI Pooling)
2. Branche de classification + régression de box (comme Faster R-CNN)
3. **Branche de masque** : FCN qui prédit un masque binaire pour chaque RoI

3.6.1 RoI Align

Problème de RoI Pooling : quantification qui cause un misalignment pixel-level.

Solution RoI Align :

- Ne pas quantifier les coordonnées de la RoI
- Utiliser une interpolation bilinéaire pour échantillonner les features
- Précision au sub-pixel level

3.6.2 Loss Function

Multi-task loss :

$$L = L_{\text{cls}} + L_{\text{box}} + L_{\text{mask}} \quad (14)$$

où :

- L_{cls} : classification loss
- L_{box} : bounding box regression loss
- L_{mask} : binary cross-entropy par pixel pour le masque

Astuce : La loss du masque est calculée uniquement pour la classe prédite, ce qui découple classification et segmentation.

TABLE 2 – Comparaison des architectures de segmentation

Modèle	Type	mIoU (%)	Application
FCN-8s	Sémantique	62.2	Scènes générales
U-Net	Sémantique	92.0	Imagerie médicale
DeepLab v3+	Sémantique	89.0	Scènes générales
Mask R-CNN	Instance	37.1	Détection + seg.

3.7 Comparaison des Architectures de Segmentation

4 Vision Transformers (ViT)

4.1 Motivation

Les CNN ont dominé la vision par ordinateur depuis 2012. Cependant, les Transformers (chapitre 08) ont révolutionné le NLP. Peut-on appliquer les Transformers à la vision ?

Défis :

- Une image 224×224 a 50,176 pixels (vs 100 tokens en NLP)
- Attention sur tous les pixels : complexité $O(n^2)$ prohibitive

4.2 Architecture Vision Transformer (ViT, 2020)

Définition : Vision Transformer

ViT découpe l'image en patches et les traite comme des tokens :

1. Diviser l'image en patches 16×16 (ou 32×32)
2. Aplatir chaque patch en vecteur
3. Embedding linéaire + positional encoding
4. Transformer encoder standard
5. Classification via un token [CLS]

4.2.1 Patchification

Pour une image $I \in \mathbb{R}^{H \times W \times C}$ et taille de patch P :

1. Nombre de patches : $N = \frac{H \cdot W}{P^2}$
2. Chaque patch : $x_p \in \mathbb{R}^{P^2 \cdot C}$ (vecteur aplati)
3. Embedding linéaire : $z_p = E \cdot x_p$ où $E \in \mathbb{R}^{D \times (P^2 \cdot C)}$

4.2.2 Architecture Complète

Algorithm 4 Vision Transformer (ViT)

Require : Image $I \in \mathbb{R}^{H \times W \times 3}$

Ensure : Prédiction de classe y

- 1 : Découper I en N patches de taille $P \times P$
 - 2 : Aplatir chaque patch : $\{x_p^1, \dots, x_p^N\}$
 - 3 : Embedding linéaire : $z_p^i = E \cdot x_p^i$ pour $i = 1, \dots, N$
 - 4 : Ajouter token [CLS] : $z_0 = z_{\text{cls}}$
 - 5 : Ajouter positional encoding : $z_i \leftarrow z_i + E_{\text{pos}}^i$
 - 6 : Passer dans L couches de Transformer Encoder
 - 7 : Extraire z_0^L (état final du token [CLS])
 - 8 : Classification : $y = \text{MLP}(z_0^L)$
 - 9 : **return** y
-

4.2.3 Positional Encoding

Contrairement au NLP, on utilise des **positional embeddings appris** :

$$E_{\text{pos}} \in \mathbb{R}^{(N+1) \times D} \quad (15)$$

Chaque position (patch) a son embedding de position appris durant l'entraînement.

Alternative : 2D positional encodings qui encodent séparément les coordonnées x et y du patch.

4.3 Variantes de ViT

4.3.1 ViT-Base, ViT-Large, ViT-Huge

TABLE 3 – Variantes de ViT

Modèle	Layers	Hidden Size	Heads
ViT-Base	12	768	12
ViT-Large	24	1024	16
ViT-Huge	32	1280	16

4.3.2 DeiT (Data-efficient image Transformers)

Améliore l'entraînement de ViT :

- **Distillation token** : apprendre d'un CNN teacher
- Augmentations agressives
- Entraînement plus efficace (moins de données)

4.3.3 Swin Transformer (2021)

Idée clé : Attention locale dans des fenêtres décalées (shifted windows).

1. Diviser l'image en fenêtres non-chevauchantes

2. Appliquer self-attention uniquement dans chaque fenêtre
3. Décaler les fenêtres entre couches pour capturer interactions cross-window
4. Hierarchical architecture (comme CNN) avec downsampling progressif

Avantages :

- Complexité linéaire : $O(H \cdot W)$ au lieu de $O((H \cdot W)^2)$
- Pyramidal features (multi-scale)
- État de l'art sur détection et segmentation

4.4 CNN vs ViT

TABLE 4 – Comparaison CNN vs ViT

Propriété	CNN	ViT
Inductive bias	Fort (localité, translation)	Faible
Données requises	Moins (ImageNet)	Plus (JFT-300M)
Précision (ImageNet)	88.5% (EfficientNet)	90.4% (ViT-Huge)
Complexité	$O(H \cdot W)$	$O((H \cdot W)^2)$
Interprétabilité	Filtres, feature maps	Attention maps

Astuce**Quand utiliser ViT ?**

- Beaucoup de données disponibles (pré-entraînement sur large dataset)
- Besoin de capturer des dépendances globales (pas seulement locales)
- Ressources GPU importantes

Quand utiliser CNN ?

- Dataset modeste ($< 100k$ images)
- Contraintes computationnelles (edge devices)
- Tâches où la localité est importante

5 Modèles Vision-Langage : CLIP**5.1 Motivation**

Les modèles de vision classiques apprennent à classifier des images en classes fixes. CLIP (Contrastive Language-Image Pre-training) apprend à associer images et textes libres.

Applications :

- Zero-shot classification : classifier sans entraînement spécifique
- Image retrieval : chercher des images par description textuelle
- Vision-language reasoning

5.2 Architecture CLIP

Définition : CLIP

CLIP entraîne conjointement un encodeur d'images et un encodeur de texte pour aligner leurs représentations dans un espace latent commun.

5.2.1 Composants

1. **Image Encoder** : ViT ou ResNet \rightarrow vecteur $v_I \in \mathbb{R}^D$
2. **Text Encoder** : Transformer \rightarrow vecteur $v_T \in \mathbb{R}^D$
3. **Contrastive Learning** : maximiser similarité cosinus pour paires correctes

5.2.2 Loss Contrastive

Pour un batch de N paires (image, texte) :

$$L = -\frac{1}{N} \sum_{i=1}^N \left[\log \frac{\exp(v_I^i \cdot v_T^i / \tau)}{\sum_{j=1}^N \exp(v_I^i \cdot v_T^j / \tau)} + \log \frac{\exp(v_I^j \cdot v_T^j / \tau)}{\sum_{j=1}^N \exp(v_I^j \cdot v_T^i / \tau)} \right] \quad (16)$$

où τ est une température apprise.

Intuition : Maximiser la similarité entre image i et texte i , minimiser avec les autres.

5.3 Zero-Shot Classification

Algorithm 5 CLIP Zero-Shot Classification

Require : Image I , classes $\{c_1, \dots, c_K\}$

Ensure : Classe prédite

- 1 : Encoder l'image : $v_I = \text{ImageEncoder}(I)$
 - 2 : **for** chaque classe c_k **do**
 - 3 : Créer prompt : $t_k = \text{"A photo of a } \{c_k\} \text{"}$
 - 4 : Encoder le texte : $v_T^k = \text{TextEncoder}(t_k)$
 - 5 : Calculer similarité : $s_k = v_I \cdot v_T^k$
 - 6 : **end for**
 - 7 : Softmax : $P(c_k) = \frac{\exp(s_k / \tau)}{\sum_j \exp(s_j / \tau)}$
 - 8 : **return** $\text{argmax}_k P(c_k)$
-

Avantage : Pas besoin de réentraîner pour de nouvelles classes, il suffit de changer les prompts !

5.4 Résultats CLIP

- Entraîné sur 400M paires (image, texte) du web
- Zero-shot sur ImageNet : 76.2% (comparable à ResNet-50 entraîné supervisé)
- Très robuste aux distribution shifts (ImageNet-A, ImageNet-R)
- Généralisation impressionnante à de nouveaux domaines

6 Implémentation

6.1 Détection d'Objets avec YOLOv8

```

1 from ultralytics import YOLO
2 import cv2
3
4 # Charger modèle pré-entraîné
5 model = YOLO('yolov8n.pt') # n, s, m, l, x
6
7 # Inférence sur une image
8 results = model('image.jpg')
9
10 # Afficher résultats
11 for r in results:
12     boxes = r.boxes # Bounding boxes
13     for box in boxes:
14         x1, y1, x2, y2 = box.xyxy[0]
15         conf = box.conf[0]
16         cls = int(box.cls[0])
17         label = model.names[cls]
18         print(f"{label} ({conf:.2f}): [{x1:.0f}, {y1:.0f}, {x2:.0f}, {y2:.0f}]")
19
20 # Entraîner sur custom dataset (COCO format)
21 model.train(
22     data='custom_dataset.yaml',
23     epochs=100,
24     imgsz=640,
25     batch=16,
26     name='yolov8_custom'
27 )
28
29 # Évaluation
30 metrics = model.val()
31 print(f"mAP50: {metrics.box.map50}")
32 print(f"mAP50-95: {metrics.box.map}")

```

Listing 1 – YOLOv8 avec Ultralytics

6.2 Segmentation avec U-Net (PyTorch)

```

1 import torch
2 import torch.nn as nn
3
4 class UNet(nn.Module):
5     def __init__(self, in_channels=3, out_channels=1):
6         super().__init__()
7
8         # Encoder
9         self.enc1 = self.conv_block(in_channels, 64)
10        self.enc2 = self.conv_block(64, 128)

```

```

11     self.enc3 = self.conv_block(128, 256)
12     self.enc4 = self.conv_block(256, 512)
13
14     # Bottleneck
15     self.bottleneck = self.conv_block(512, 1024)
16
17     # Decoder
18     self.upconv4 = nn.ConvTranspose2d(1024, 512, 2, stride=2)
19     self.dec4 = self.conv_block(1024, 512)
20
21     self.upconv3 = nn.ConvTranspose2d(512, 256, 2, stride=2)
22     self.dec3 = self.conv_block(512, 256)
23
24     self.upconv2 = nn.ConvTranspose2d(256, 128, 2, stride=2)
25     self.dec2 = self.conv_block(256, 128)
26
27     self.upconv1 = nn.ConvTranspose2d(128, 64, 2, stride=2)
28     self.dec1 = self.conv_block(128, 64)
29
30     # Output
31     self.out = nn.Conv2d(64, out_channels, 1)
32
33     self.pool = nn.MaxPool2d(2)
34
35     def conv_block(self, in_ch, out_ch):
36         return nn.Sequential(
37             nn.Conv2d(in_ch, out_ch, 3, padding=1),
38             nn.BatchNorm2d(out_ch),
39             nn.ReLU(inplace=True),
40             nn.Conv2d(out_ch, out_ch, 3, padding=1),
41             nn.BatchNorm2d(out_ch),
42             nn.ReLU(inplace=True)
43         )
44
45     def forward(self, x):
46         # Encoder
47         enc1 = self.enc1(x)
48         enc2 = self.enc2(self.pool(enc1))
49         enc3 = self.enc3(self.pool(enc2))
50         enc4 = self.enc4(self.pool(enc3))
51
52         # Bottleneck
53         bottleneck = self.bottleneck(self.pool(enc4))
54
55         # Decoder with skip connections
56         dec4 = self.upconv4(bottleneck)
57         dec4 = torch.cat([dec4, enc4], dim=1)
58         dec4 = self.dec4(dec4)
59
60         dec3 = self.upconv3(dec4)
61         dec3 = torch.cat([dec3, enc3], dim=1)

```

```

62         dec3 = self.dec3(dec3)
63
64         dec2 = self.upconv2(dec3)
65         dec2 = torch.cat([dec2, enc2], dim=1)
66         dec2 = self.dec2(dec2)
67
68         dec1 = self.upconv1(dec2)
69         dec1 = torch.cat([dec1, enc1], dim=1)
70         dec1 = self.dec1(dec1)
71
72         return self.out(dec1)
73
74 # Entraînement
75 model = UNet(in_channels=3, out_channels=1).cuda()
76 criterion = nn.BCEWithLogitsLoss()
77 optimizer = torch.optim.Adam(model.parameters(), lr=1e-4)
78
79 for epoch in range(epochs):
80     for images, masks in train_loader:
81         images, masks = images.cuda(), masks.cuda()
82
83         outputs = model(images)
84         loss = criterion(outputs, masks)
85
86         optimizer.zero_grad()
87         loss.backward()
88         optimizer.step()

```

Listing 2 – U-Net implémentation

6.3 Vision Transformer avec timm

```

1 import timm
2 import torch
3
4 # Charger modèle pré-entraîné
5 model = timm.create_model('vit_base_patch16_224', pretrained=True)
6
7 # Voir tous les modèles ViT disponibles
8 vit_models = timm.list_models('vit*')
9 print(f"Modèles ViT disponibles: {len(vit_models)}")
10
11 # Fine-tuning sur dataset custom
12 model = timm.create_model('vit_base_patch16_224',
13                           pretrained=True,
14                           num_classes=10)
15
16 # Geler l'encoder, entraîner seulement la tête
17 for param in model.parameters():
18     param.requires_grad = False
19 for param in model.head.parameters():

```

```

20     param.requires_grad = True
21
22 # Entraînement
23 optimizer = torch.optim.AdamW(model.head.parameters(), lr=1e-3)
24 criterion = torch.nn.CrossEntropyLoss()
25
26 model.train()
27 for images, labels in train_loader:
28     outputs = model(images)
29     loss = criterion(outputs, labels)
30
31     optimizer.zero_grad()
32     loss.backward()
33     optimizer.step()
34
35 # Visualiser attention maps
36 attention_weights = model.blocks[-1].attn.get_attention_map()

```

Listing 3 – ViT avec timm

6.4 CLIP Zero-Shot

```

1 import torch
2 import clip
3 from PIL import Image
4
5 # Charger modèle CLIP
6 device = "cuda" if torch.cuda.is_available() else "cpu"
7 model, preprocess = clip.load("ViT-B/32", device=device)
8
9 # Préparer image
10 image = preprocess(Image.open("cat.jpg")).unsqueeze(0).to(device)
11
12 # Définir classes possibles
13 text_prompts = [
14     "a photo of a cat",
15     "a photo of a dog",
16     "a photo of a bird",
17     "a photo of a car"
18 ]
19 text = clip.tokenize(text_prompts).to(device)
20
21 # Inférence
22 with torch.no_grad():
23     image_features = model.encode_image(image)
24     text_features = model.encode_text(text)
25
26 # Normaliser
27 image_features /= image_features.norm(dim=-1, keepdim=True)
28 text_features /= text_features.norm(dim=-1, keepdim=True)
29

```

```
30     # Calculer similarités
31     similarity = (100.0 * image_features @ text_features.T).softmax(dim
32     =-1)
33     values, indices = similarity[0].topk(4)
34 # Résultats
35 for value, index in zip(values, indices):
36     print(f"{text_prompts[index]:20s}: {100 * value.item():.2f}%")
```

Listing 4 – CLIP avec OpenAI

7 Avantages et Limites

7.1 Object Detection

7.1.1 Avantages

- Localisation précise des objets dans l'image
- Performance temps réel avec YOLO (vidéo, applications embarquées)
- Datasets annotés disponibles (COCO, Pascal VOC)
- Transfert d'apprentissage efficace

7.1.2 Limites

- Annotation coûteuse (bounding boxes pour chaque objet)
- Difficulté avec objets très petits ou occultés
- Trade-off précision vs vitesse
- Sensible aux variations d'échelle

7.2 Segmentation

7.2.1 Avantages

- Précision au pixel près (vs boxes rectangulaires)
- U-Net très efficace en imagerie médicale
- Mask R-CNN : détection + segmentation en un modèle
- Interprétabilité : visualiser exactement les régions d'intérêt

7.2.2 Limites

- Annotation pixel-level extrêmement coûteuse
- Plus lent que la détection (surtout segmentation d'instances)
- Difficulté avec les frontières ambiguës
- Mémoire GPU importante

7.3 Vision Transformers

7.3.1 Avantages

- Capture des dépendances globales (pas seulement locales comme CNN)

- Meilleure précision avec beaucoup de données
- Attention maps interprétables
- Architecture unifiée pour vision et NLP

7.3.2 Limites

- Requiert énormément de données (JFT-300M, ImageNet-21k)
- Complexité quadratique en la taille de l'image
- Moins bon que CNN avec peu de données
- Coût computationnel élevé

7.4 CLIP

7.4.1 Avantages

- Zero-shot learning : pas besoin de réentraîner pour nouvelles classes
- Robuste aux distribution shifts
- Multimodal : image + texte
- Flexibilité : prompts textuels adaptables

7.4.2 Limites

- Performances inférieures au fine-tuning sur tâches spécifiques
- Sensible à la formulation des prompts
- Biais des données web (400M paires non filtrées)
- Difficulté avec tâches nécessitant localisation précise

8 Hyperparamètres et Tuning

8.1 Détection d'Objets

TABLE 5 – Hyperparamètres clés pour la détection

Paramètre	Valeurs typiques	Impact
learning_rate	10^{-4} à 10^{-2}	Convergence
image_size	416, 640, 1024	Précision vs vitesse
batch_size	8 à 32	Stabilité gradient
IoU_threshold	0.5 à 0.7	NMS agressivité
conf_threshold	0.25 à 0.5	Rappel vs précision
anchor_scales	Auto ou manuel	Détection multi-échelle

TABLE 6 – Hyperparamètres clés pour la segmentation

Paramètre	Valeurs typiques	Impact
learning_rate	10^{-4} à 10^{-3}	Convergence
batch_size	2 à 16	Mémoire GPU
image_size	256, 512, 1024	Détails vs vitesse
encoder_depth	4 à 5	Complexité modèle
dropout	0.1 à 0.5	Régularisation
class_weights	Auto ou manuel	Déséquilibre classes

TABLE 7 – Hyperparamètres clés pour ViT

Paramètre	Valeurs typiques	Impact
patch_size	16, 32	Nb tokens vs détails
num_layers	12 à 32	Capacité modèle
hidden_dim	768 à 1280	Capacité représentation
num_heads	12 à 16	Attention multi-échelle
learning_rate	10^{-4} à 10^{-3}	Convergence
warmup_steps	500 à 10000	Stabilité initiale

8.2 Segmentation

8.3 Vision Transformers

Astuce

Stratégies d'entraînement efficaces :

- **Transfer learning** : Toujours partir d'un modèle pré-entraîné (ImageNet, COCO)
- **Progressive resizing** : Commencer avec petites images, augmenter progressivement
- **Mixed precision** : FP16 pour accélérer et économiser mémoire
- **Data augmentation** : Mosaic, MixUp, CutMix pour la détection
- **Learning rate schedule** : Cosine annealing ou OneCycleLR

9 Applications Pratiques

9.1 Conduite Autonome

- **Détection** : Voitures, piétons, vélos, panneaux de signalisation
- **Segmentation sémantique** : Route, trottoir, végétation, bâtiments
- **Segmentation d'instances** : Suivi de chaque voiture/piéton individuellement
- **Modèles** : YOLOv8, Mask R-CNN, DeepLab v3+
- **Datasets** : KITTI, Cityscapes, nuScenes

9.2 Imagerie Médicale

- **Segmentation d'organes** : U-Net pour foie, reins, cerveau
- **Détection de tumeurs** : Faster R-CNN, RetinaNet
- **Segmentation cellulaire** : U-Net, Mask R-CNN

- **Modèles** : U-Net (référence), nnU-Net (auto-configuration)
- **Datasets** : Medical Segmentation Decathlon, LIDC-IDRI

9.3 Reconnaissance Faciale

- **Détection de visages** : MTCNN, RetinaFace
- **Landmarks faciaux** : 68 points de repère (yeux, nez, bouche)
- **Segmentation** : Cheveux, peau, arrière-plan
- **Applications** : Sécurité, filtres AR, analyse d'émotions

9.4 Commerce Électronique

- **Recherche visuelle** : CLIP pour "trouver des produits similaires"
- **Détection de produits** : Compter articles en rayon (retail)
- **Segmentation produits** : Extraction de produit pour montage
- **Modèles** : CLIP, YOLOv8, Mask R-CNN

9.5 Surveillance et Sécurité

- **Détection d'intrusion** : YOLO temps réel
- **Suivi multi-objets** : DeepSORT avec détecteur
- **Détection d'anomalies** : Comportements inhabituels
- **Comptage de personnes** : Segmentation + tracking

10 Résumé du Chapitre

10.1 Points Clés

- **Object Detection** : Localiser et classifier des objets
 - Two-stage : Faster R-CNN (précis, 5 FPS)
 - One-stage : YOLO (rapide, 80 FPS)
 - Métriques : IoU, mAP@0.5, mAP@0.5 : 0.95
- **Segmentation Sémantique** : Classifier chaque pixel
 - FCN : première architecture fully convolutional
 - U-Net : architecture emblématique (médical)
 - DeepLab : atrous convolutions + ASPP
 - Métriques : mIoU, Dice coefficient
- **Segmentation d'Instances** : Détecter et segmenter chaque instance
 - Mask R-CNN = Faster R-CNN + branche de masque
 - RoI Align pour précision pixel-level
- **Vision Transformers** : Appliquer les Transformers à la vision
 - ViT : découper en patches, self-attention
 - Swin Transformer : attention locale + shifted windows
 - Nécessite beaucoup de données
- **Vision-Langage (CLIP)** : Aligner images et textes
 - Contrastive learning sur 400M paires
 - Zero-shot classification par prompts

— Applications : recherche, multimodal

10.2 Formules Essentielles

Formules à retenir

IoU (Intersection over Union) :

$$\text{IoU} = \frac{\text{Aire}(A \cap B)}{\text{Aire}(A \cup B)}$$

Mean Average Precision :

$$\text{mAP} = \frac{1}{C} \sum_{c=1}^C AP_c$$

Dice Coefficient :

$$\text{Dice} = \frac{2 \cdot TP}{2 \cdot TP + FP + FN}$$

ViT Patch Embedding :

$$z_p = E \cdot x_p + E_{pos}, \quad x_p \in \mathbb{R}^{P^2 \cdot C}$$

CLIP Contrastive Loss :

$$L = -\log \frac{\exp(v_I \cdot v_T / \tau)}{\sum_j \exp(v_I \cdot v_T^j / \tau)}$$

10.3 Tableau Récapitulatif

TABLE 8 – Comparaison des approches de vision avancée

Tâche	Modèle Recommandé	Précision	Vitesse
Détection (précision)	Faster R-CNN		
Détection (temps réel)	YOLOv8		
Seg. sémantique	DeepLab v3+		
Seg. médicale	U-Net		
Seg. instances	Mask R-CNN		
Classification	ViT-Large		
Zero-shot	CLIP		

11 Exercices

11.1 Questions de Compréhension

1. Expliquer la différence entre R-CNN, Fast R-CNN et Faster R-CNN. Pourquoi chaque version est-elle plus rapide que la précédente ?
2. Pourquoi YOLO est-il appelé "You Only Look Once" ? Quelle est sa principale différence philosophique avec Faster R-CNN ?
3. Qu'est-ce que le RoI Pooling ? Pourquoi RoI Align est-il meilleur pour la segmentation ?

4. Expliquer le rôle des skip connections dans U-Net. Pourquoi utiliser la concaténation plutôt que l'addition ?
5. Comment fonctionne l'atrous convolution dans DeepLab ? Quel est son avantage ?
6. Pourquoi ViT nécessite-t-il beaucoup plus de données d'entraînement que les CNN ?
7. Comment CLIP permet-il la classification zero-shot ? Donner un exemple d'application.
8. Quelle est la différence entre segmentation sémantique et segmentation d'instances ?

11.2 Exercices Pratiques

1. **Détection avec YOLOv8**
 - Entraîner YOLOv8 sur un dataset custom (par ex. détecter des visages)
 - Tester sur vidéo et mesurer le FPS
 - Comparer YOLOv8n (nano) vs YOLOv8x (extra-large) : précision vs vitesse
 - Notebook : `12demoobject_detection.ipynb`
2. **Segmentation médicale avec U-Net**
 - Implémenter U-Net from scratch en PyTorch
 - Entraîner sur un dataset de segmentation (par ex. cellules, tumeurs)
 - Calculer Dice coefficient et mIoU
 - Visualiser les prédictions
 - Notebook : `12demosegmentation.ipynb`
3. **Vision Transformers**
 - Fine-tuner ViT-Base sur CIFAR-10
 - Comparer avec ResNet-50
 - Visualiser les attention maps
 - Tester DeiT avec distillation
 - Notebook : `12demo_vision_transformers.ipynb`
4. **CLIP Zero-Shot**
 - Utiliser CLIP pour classifier des images sans fine-tuning
 - Tester différents prompts et mesurer l'impact
 - Implémenter un moteur de recherche d'images par description textuelle

12 Pour Aller Plus Loin

12.1 Lectures Recommandées

12.1.1 Papers Fondateurs

- **R-CNN** : Girshick et al. (2014). "Rich feature hierarchies for accurate object detection and semantic segmentation"
- **Faster R-CNN** : Ren et al. (2015). "Faster R-CNN : Towards Real-Time Object Detection with Region Proposal Networks"
- **YOLO** : Redmon et al. (2016). "You Only Look Once : Unified, Real-Time Object Detection"
- **U-Net** : Ronneberger et al. (2015). "U-Net : Convolutional Networks for Biomedical Image Segmentation"
- **Mask R-CNN** : He et al. (2017). "Mask R-CNN"

- **ViT** : Dosovitskiy et al. (2020). "An Image is Worth 16x16 Words : Transformers for Image Recognition at Scale"
- **CLIP** : Radford et al. (2021). "Learning Transferable Visual Models From Natural Language Supervision"

12.1.2 Papers Récents

- **YOLOv8** : Ultralytics (2023). Documentation officielle
- **Swin Transformer** : Liu et al. (2021). "Swin Transformer : Hierarchical Vision Transformer using Shifted Windows"
- **Segment Anything (SAM)** : Kirillov et al. (2023). "Segment Anything"
- **DINOv2** : Oquab et al. (2023). "DINOv2 : Learning Robust Visual Features without Supervision"

12.2 Ressources en Ligne

- Ultralytics YOLOv8 : <https://docs.ultralytics.com/>
- Detectron2 (Facebook) : <https://github.com/facebookresearch/detectron2>
- MMDetection : <https://github.com/open-mmlab/mmdetection>
- Timm (PyTorch Image Models) : <https://github.com/huggingface/pytorch-image-models>
- OpenAI CLIP : <https://github.com/openai/CLIP>
- Hugging Face Transformers : <https://huggingface.co/docs/transformers/>

12.3 Datasets

- COCO (detection, segmentation) : <https://cocodataset.org/>
- Pascal VOC : <http://host.robots.ox.ac.uk/pascal/VOC/>
- Cityscapes (conduite) : <https://www.cityscapes-dataset.com/>
- Medical Segmentation Decathlon : <http://medicaldecathlon.com/>
- Open Images : <https://storage.googleapis.com/openimages/web/index.html>

12.4 Outils et Bibliothèques

- Ultralytics : YOLO v5/v8 (PyTorch)
- torchvision : Modèles pré-entraînés (Faster R-CNN, Mask R-CNN)
- segmentation_models.pytorch : U-Net, DeepLab, etc.
- timm : Vision Transformers
- transformers : CLIP, ViT (Hugging Face)
- albumentations : Augmentations avancées

12.5 Prochaines Étapes

- Chapitre 14 - GANs et Modèles Génératifs : StyleGAN, Diffusion Models
- Chapitre 15 - AutoML et NAS : Architecture search automatique
- Projets pratiques : Appliquer ces techniques à vos propres données

Références

1. Girshick, R., Donahue, J., Darrell, T., & Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. *CVPR*.
2. Ren, S., He, K., Girshick, R., & Sun, J. (2015). Faster R-CNN : Towards Real-Time Object Detection with Region Proposal Networks. *NeurIPS*.
3. Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You Only Look Once : Unified, Real-Time Object Detection. *CVPR*.
4. Ronneberger, O., Fischer, P., & Brox, T. (2015). U-Net : Convolutional Networks for Biomedical Image Segmentation. *MICCAI*.
5. He, K., Gkioxari, G., Dollár, P., & Girshick, R. (2017). Mask R-CNN. *ICCV*.
6. Chen, L. C., Papandreou, G., Schroff, F., & Adam, H. (2017). Rethinking Atrous Convolution for Semantic Image Segmentation. *arXiv*.
7. Dosovitskiy, A., et al. (2020). An Image is Worth 16x16 Words : Transformers for Image Recognition at Scale. *ICLR 2021*.
8. Liu, Z., et al. (2021). Swin Transformer : Hierarchical Vision Transformer using Shifted Windows. *ICCV*.
9. Radford, A., et al. (2021). Learning Transferable Visual Models From Natural Language Supervision. *ICML*.
10. Kirillov, A., et al. (2023). Segment Anything. *ICCV*.