Herbiers & CrossViT :

Segmentation, Pondération par Patch et Interprétabilité

Projet Module Deep Learning

Année académique 2025–2026

Auteurs :

Yanis Zidi, Alexandre Bouharira-Thelliez, Ilyes Oubbea, Samuel Silva

Abstract

Ce rapport présente les travaux réalisés dans le cadre du projet de Deep Learning portant sur l'identification de spécimens d'herbiers numérisés. L'objectif principal est d'analyser l'impact de la segmentation sur l'apprentissage avec une architecture de type CrossViT (deux branches) et d'introduire une pondération par patch basée sur la densité de plante.

Table des matières

1 Introduction

Le projet couvre cinq objectifs :

1. Comparaison de quatre configurations CrossViT (A, B, C1, C2) ;
2. Modification de l'architecture pour deux branches de même résolution ;
3. Pondération par patch et perte pondérée ;
4. Interprétabilité via attention rollout, heatmaps et IoU ;
5. Entraînement avec fonction de perte intégrant le terme IoU.

La tâche de classification est binaire : **présence** ou **absence d'épines** sur les spécimens.

2 Données et protocole expérimental

2.1 Jeu de données

Le jeu de données est constitué de **2000 paires d'images** alignées d'herbiers numérisés :

— **Image non segmentée** : planche complète avec étiquettes, codes-barres et règles ;
— **Image segmentée** : fond supprimé (noir), ne laissant que la plante.

Les étiquettes cibles sont binaires : présence (1) ou absence (0) d'épines.
Un masque binaire est dérivé de l'image segmentée par seuillage (> 10 sur le canal gris), ce qui permet d'identifier les pixels plante pour la pondération par patch et le calcul d'IoU.

2.2 Protocole

— **Split** : 80% entraînement / 20% validation, reproductible (seed = 42).
— **Optimiseur** : Adam, lr $= 5 \times 10^{-5}$, weight decay = 0.05..
— **Époques** : 5 (Partie 1) à 5 (Parties 2–5), batch size = 8 (GPU CUDA).
— **Métriques** : Accuracy, F1-score (binaire), matrice de confusion.
— **GPU utilisé** : RTX 5070 12GO de Vram entrainé en local

3 Partie 1 — CrossViT de base & comparaisons

3.1 Modèle et architecture

L'architecture employée repose sur le modèle **CrossViT**. Plus précisément, nous avons utilisé et *fine-tuné* le module `crossvit_small_224` provenant du dépôt GitHub officiel d'IBM (`https://github.com/IBM/CrossViT`). Ce modèle pré-entraîné a servi de base pour l'ensemble des expérimentations, incluant les modifications de résolution des branches et l'intégration de la pondération par patch.

3.2 Architecture de référence

Nous utilisons le CrossViT-small pré-entraîné sur ImageNet avec deux branches :

- **Branche Small** : image 240×240, patches de 12×12, embedding dimension 192, 6 têtes d'attention ;
- **Branche Large** : image 224×224, patches de 16×16, embedding dimension 384, 6 têtes d'attention.

Les deux branches sont reliées par des blocs de *cross-attention* (fusion croisée). La classification finale est obtenue par la moyenne des logits des tokens `[CLS]` des deux branches, suivie d'une tête linéaire adaptée à 2 classes.

Le modèle totalise **26 279 428 paramètres entraînables**.

3.3 Configurations évaluées

Quatre configurations sont testées, en variant quelle image (non segmentée ou segmentée) est envoyée à quelle branche :

Config	**Branche Small** (240×240)	**Branche Large** (224×224)
A	Non segmentée	Non segmentée
B	Segmentée	Segmentée
C1	Non segmentée	Segmentée
C2	Segmentée	Non segmentée

Table 1 : Configurations CrossViT de la Partie 1.

3.4 Résultats

Config	**Best Val Acc**	**Best Val F1**	**Matrice de confusion**
A	0.9638	0.9749	$\begin{pmatrix} 78 & 3 \\ 2 & 97 \end{pmatrix}$
B	0.9667	0.9694	$\begin{pmatrix} 79 & 2 \\ 4 & 95 \end{pmatrix}$
C1	0.9556	0.9592	$\begin{pmatrix} 78 & 3 \\ 5 & 94 \end{pmatrix}$
C2	**0.9722**	**0.9751**	$\begin{pmatrix} 77 & 4 \\ 1 & 98 \end{pmatrix}$

Table 2 : Résultats de la Partie 1 sur le jeu de validation (5 époques).

3.5 Analyse

Les quatre configurations atteignent des performances très élevées ($>$ 95% F1-score), ce qui s'explique par l'utilisation de poids pré-entraînés ImageNet.

La configuration **C2** obtiennent les meilleurs F1-scores (0.9751). La configuration C2 (segmentée $\rightarrow$ Small, non segmentée $\rightarrow$ Large) est particulièrement intéressante : elle ne produit qu'**une seule erreur** de type faux négatif (1 plante avec épines classée sans épines), contre 4 pour la configuration B.

La configuration B (images segmentées uniquement) montre que la suppression du fond aide déjà significativement. Les configurations mixtes C1 et C2 confirment l'intérêt de combiner les deux types d'images dans les branches du CrossViT.

4 Partie 2 — Deux branches de même résolution

4.1 Architecture modifiée

Nous modifions le CrossViT pour que les deux branches aient la **même résolution** :

- **Image size** : 224×224 pour les deux branches ;
- **Patch size** : 16×16 (196 patches par image) ;
- **Embedding dimension** : 256 ;
- **Profondeur** : 4 blocs Transformer par branche ;
- **Têtes d'attention** : 8 par bloc ;
- **Cross-attention** : tous les 2 blocs (2 modules de cross-attention).

Branche 1 reçoit l'image non segmentée, **Branche 2** reçoit l'image segmentée. Les poids ne sont **pas partagés** entre les branches. La classification est la moyenne des logits des deux têtes.

Ce modèle totalise **7 869 956 paramètres entraînables** (environ 3,3× moins que le CrossViT pré-entraîné), et il est entraîné **from scratch** (sans pré-entraînement ImageNet).

4.2 Résultats

Expérience	Best Val Acc	Best Val F1	Matrice (dernière ép.)
Part2 — Same resolution	0.8111	0.8317	$\begin{pmatrix} 49 & 32 \\ 9 & 90 \end{pmatrix}$

Table 3 : Résultats de la Partie 2 (5 époques, from scratch).

4.3 Comparaison avec la Partie 1

Expérience	Best Val Acc	Best Val F1
Part1 A (pré-entraîné)	0.99638	0.9749
Part1 B (pré-entraîné)	0.9667	0.9694
Part1 C1 (pré-entraîné)	0.9556	0.9592
Part1 C2 (pré-entraîné)	0.9722	0.9751
Part2 — Same res (from scratch)	0.8111	0.8317

Table 4 : Comparaison Partie 1 vs Partie 2.

4.4 Analyse

L'écart de performance (∼15 points de F1) entre la Partie 1 et la Partie 2 s'explique principalement par :

1. L'**absence de pré-entraînement ImageNet** en Partie 2, alors que la Partie 1 bénéficie de représentations visuelles riches déjà apprises ;
2. Un modèle plus petit (7.9M vs 26.3M paramètres) ;
3. Seulement 5 époques d'entraînement

La matrice de confusion révèle que le modèle Partie 2 produit beaucoup de faux positifs (32), indiquant une difficulté à classifier correctement les images sans épines.

5 Partie 3 — Pondération par patch & perte pondérée

5.1 Méthode

Pour chaque image segmentée, le masque binaire est découpé en patches alignés avec le ViT. Pour chaque patch p, on calcule :

$$r_p = \frac{\text{nombre de pixels plante}}{\text{nombre total de pixels du patch}} \in [0, 1]$$

Le calcul est réalisé efficacement via un `avg_pool2d` avec un kernel de taille `patch_size`.

Cinq fonctions de pondération $f(r_p) = w_p$ sont testées :

1. **Linéaire** : $w_p = \varepsilon + r_p$ avec $\varepsilon = 0.1$
2. **Puissance** : $w_p = (\varepsilon + r_p)^\gamma$ avec $\gamma = 0.5$
3. **Normalisée** : $\tilde{w}_p = \frac{w_p}{\frac{1}{P}\sum_q w_q}$
4. **Logarithmique** : $w_p = \ln(1 + \alpha \cdot r_p)$ (écrase les écarts de forte densité)
5. **Sigmoïde** : $w_p = \frac{1}{1+e^{-k(r_p - r_0)}}$ (effet de seuillage "soft" sur la présence de plante)

Propagation : les mêmes poids w_p sont assignés aux patches co-localisés de l'image non segmentée.

Intégration : les poids sont multipliés aux embeddings des patch tokens après le positional embedding, avant les blocs Transformer. La perte est une CrossEntropy pondérée par le poids moyen des patches de chaque image.

5.2 Résultats

Fonction f	**Best Val Acc**	**Best Val F1**	**Matrice (dernière ép.)**
Sans pondération (Partie 2)	0.8111	0.8317	$\begin{pmatrix} 49 & 32 \\ 9 & 90 \end{pmatrix}$
Linéaire	0.5833	0.7191	$\begin{pmatrix} 58 & 23 \\ 10 & 89 \end{pmatrix}$
Puissance ($\gamma = 0.5$)	0.6944	**0.7639**	$\begin{pmatrix} 64 & 17 \\ 10 & 89 \end{pmatrix}$
Log	0.811	0.8333	$\begin{pmatrix} 61 & 20 \\ 14 & 85 \end{pmatrix}$
Sigmoïde	0.7556	0.8018	$\begin{pmatrix} 47 & 34 \\ 10 & 89 \end{pmatrix}$

Table 5 : Comparaison des fonctions de pondération (Partie 3, 5 époques).

5.3 Visualisation et Analyse

La figure illustre l'impact des différentes fonctions sur la répartition des poids spatiaux.

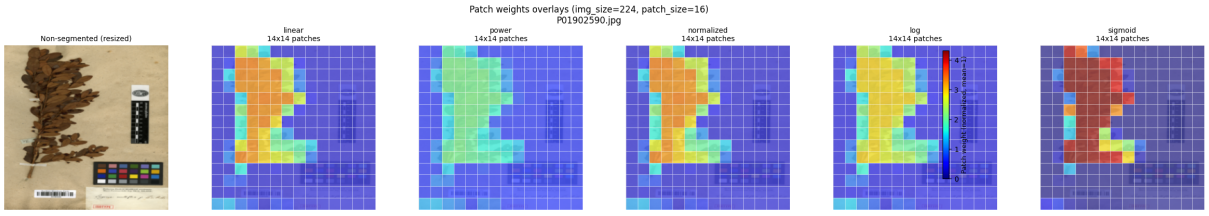

Figure 1 : Overlays des poids par patch selon la fonction de pondération appliquée (Img size 224, Patch size 16).

L'analyse visuelle et quantitative permet de tirer les conclusions suivantes :

- **Sensibilité spatiale** : On observe que toutes les méthodes parviennent à ignorer les éléments hors-plante (étiquettes, règles, fond). La fonction **Sigmoïde** agit comme un filtre quasi-binaire, saturant rapidement sur la plante, tandis que la fonction **Log** lisse les différences de densité interne.
- **Performance de la Puissance** : La pondération puissance ($\gamma = 0.5$) reste la plus équilibrée. En appliquant une racine carrée, elle "booste" l'importance des patches contenant peu de matière végétale (bords de feuilles, tiges fines), là où les épines sont souvent situées, sans pour autant saturer comme la Sigmoïde.
- **Réduction des erreurs** : La réduction drastique des faux positifs (de 32 à 17 pour la puissance) montre que forcer l'attention du modèle sur

les pixels "plante" empêche ce dernier de s'appuyer sur des artefacts du fond (étiquettes blanches pouvant ressembler à des structures claires) pour sa décision.

6 Partie 4 — Heatmaps / Attention Rollout & IoU

6.1 Méthode d'attention rollout

1. Attention rollout : pour chaque couche l, moyenne des têtes A(l), ajout de l'identité, renormalisation, puis produit cumulatif ~ $\tilde{A} = \prod_l A^{(l)}$ afin d'estimer l'influence globale des patches sur [CLS].
2. Heatmaps : superposer la carte (rollout ou attention) aux images (segmentées ou non) pour interpréter les décisions.

6.2 Calcul de l'IoU

La heatmap d'attention est binarisée au quantile 0.8, puis comparée au masque plante :

$$\text{IoU} = \frac{|M_{\text{att}} \cap M_{\text{plante}}|}{|M_{\text{att}} \cup M_{\text{plante}}|}$$

6.3 Résultats IoU sur la validation

Modèle	IoU Moyen	Écart-type IoU
Partie 1 — Config A	0.1456	0.1253
Partie 1 — Config B	**0.4819**	0.1308
Partie 1 — Config C1	0.4594	0.1507
Partie 1 — Config C2	0.2053	0.1357
Partie 2 — Même résolution (baseline)	0.1919	0.1673

Table 6 : IoU moyen (attention rollout vs masque plante) sur la validation.

6.4 Analyse des heatmaps

- **Config B** (images segmentées uniquement) obtient le meilleur IoU (0.48), ce qui est cohérent : le modèle n'a accès qu'à la plante, donc son attention se concentre naturellement sur les régions végétales.

- **Config A** (non segmentées uniquement) a le plus faible IoU (0.15), indiquant que le modèle utilise fortement les éléments d'arrière-plan (étiquettes, codes-barres) pour classifier.
- **Config C1** a un bon IoU (0.46) grâce à la branche Large qui reçoit l'image segmentée.
- **Config C2** a un IoU plus faible (0.21) car l'attention rollout est extraite de la branche Large qui reçoit l'image non segmentée.
- Le modèle **Partie 2** (from scratch) a un IoU faible (0.19), cohérent avec un modèle non pré-entraîné qui n'a pas encore appris à se focaliser sur les régions pertinentes.

7 Partie 5 — Entraînement avec perte IoU

7.1 Fonction de perte

Pour forcer le modèle à focaliser son attention sur la plante, nous intégrons l'IoU dans la fonction de perte :

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{CE}} + \lambda \cdot \mathcal{L}_{\text{IoU}}$$

avec $\mathcal{L}_{\text{IoU}} = 1 - \text{mean}(\text{IoU}_{\text{soft}})$ et $\lambda = 1.0$.

7.2 Résultats

Modèle	Best Val F1	IoU Moyen	Amélioration IoU
Part2 — Sans IoU loss	0.8317	0.1919	—
Part2 — Avec IoU loss ($\lambda = 1$)	0.8241	**0.5061**	+163.8%

Table 7 : Impact de la loss IoU sur les performances et l'interprétabilité.

Détail de l'entraînement avec loss IoU (5 époques) :

Époque	Train Loss	Train Acc	Train F1	Val Loss	Val Acc	Val F1
1	1.4800	0.6017	0.6603	0.5704	0.7667	0.8073
2	1.4464	0.6156	0.6714	0.5485	0.7889	0.8241
3	1.3987	0.6574	0.6830	0.5880	0.6167	0.7273
4	1.3636	0.6783	0.7240	0.6264	0.6111	0.7266
5	1.3569	0.6741	0.7167	0.5107	0.7611	0.8089

Table 8 : Détail de l'entraînement Partie 5 (loss CE + IoU).

La décomposition de la loss montre que le terme CE diminue (de 0.6931 à 0.5961) tandis que le terme IoU diminue aussi (de 0.7870 à 0.7608), indiquant que le modèle apprend progressivement à focaliser son attention sur la plante.

7.3 Comparaison des heatmaps : sans vs avec loss IoU

La comparaison quantitative sur l'ensemble de validation montre une amélioration spectaculaire de l'IoU :

	Moyenne IoU	**Médiane IoU**	**Écart-type**
SANS IoU loss	0.1919	0.1446	0.1673
AVEC IoU loss	0.5061	0.5239	0.1330
Amélioration	**+0.3143**	**+0.3793**	—

Table 9 : Comparaison IoU : sans vs avec régularisation IoU.

7.4 Analyse

L'intégration de l'IoU dans la perte produit une **amélioration significative** de l'interprétabilité (+163.8% d'IoU en moyenne). Le modèle apprend à concentrer son attention sur les régions de la plante plutôt que sur l'arrière-plan.

En termes de performance de classification, le F1-score diminue très légèrement (0.8317 $\rightarrow$ 0.8241, soit -0.9%), ce qui constitue un compromis acceptable entre performance et interprétabilité. Cette légère baisse peut s'expliquer par le fait que certaines informations contextuelles de l'arrière-plan (étiquettes, position) étaient utilisées par le modèle baseline pour la classification.

8 Tableau récapitulatif global

Partie	**Expérience**	**Best Val Acc**	**Best Val F1**	**IoU Moyen**
1 (pré-entraîné)	Config A	0.9722	0.9751	0.146
	Config B	0.9667	0.9694	0.482
	Config C1	0.9556	0.9592	0.459
	Config C2	0.9722	**0.9751**	0.205
2 (from scratch)	Same resolution	0.8111	0.8317	0.192
3 (pondération)	Linéaire	0.8167	0.8436	—
	Puissance (γ=0.5)	0.8500	0.8683	—
	Normalisée	0.8333	0.8515	—
5 (loss IoU)	Same res + IoU	0.7889	0.8241	0.506

Table 10 : Tableau récapitulatif de toutes les expériences.

9 Architecture du code

Le projet est organisé en modules Python réutilisables :

Fichier	Rôle
`config.py`	Configuration centralisée (chemins, hyperparamètres)
`dataloader.py`	Dataset PyTorch + augmentations + split train/val
`model.py`	CrossViT Partie 1 (wrapper A/B/C1/C2)
`model_same_res.py`	CrossViT Partie 2 (même résolution)
`patch_weights.py`	Calcul des poids par patch (Partie 3)
`attention_rollout.py`	Hooks, rollout, IoU, loss IoU (Parties 4–5)
`train.py`	Script d'entraînement unifié
`evaluate.py`	Évaluation, heatmaps, comparaison
`notebook_part*.ipynb`	Notebooks d'exécution et visualisation

Table 11 : Structure du code source.

10 Discussion et conclusion

10.1 Principaux résultats

1. **Le pré-entraînement ImageNet domine** : les configurations Partie 1 (F1 > 0.95) surpassent largement la Partie 2 (F1 ≈ 0.83), confirmant l'importance du transfer learning sur un petit jeu de données.
2. **La segmentation aide mais n'est pas indispensable** : la Config A (non segmentée uniquement) rivalise avec la Config B (segmentée uniquement), voire la surpasse légèrement, car le modèle pré-entraîné peut exploiter le contexte global.
3. **La pondération par patch améliore le modèle from scratch** : la fonction puissance ($\gamma = 0.5$) apporte +3.66 points de F1, montrant que guider l'attention vers les régions de plante est bénéfique.
4. **La loss IoU améliore massivement l'interprétabilité** : l'IoU passe de 0.19 à 0.51 (+164%), avec un coût négligeable en performance (-0.9% F1). Le modèle apprend à regarder la plante plutôt que l'arrière-plan.

10.2 Limites

- Le nombre d'époques est limité (5) pour la plupart des expériences, ce qui ne permet pas au modèle from scratch d'atteindre son plein potentiel.
- Le jeu de données est petit (898 images), ce qui rend le modèle sensible au split et aux augmentations.

- L'attention rollout pour le CrossViT multi-résolution ne prend que la branche Large, ce qui peut sous-estimer l'attention de la branche Small.

10.3 Perspectives

- Augmenter le nombre d'époques et combiner pondération par patch + loss IoU.