



## RECONNAISSANCE DE MAUVAISES HERBES

Projet de deuxième année

CALIO Hugo, BLASCO Stanislas

Encadrant : M. CHAPRON  
2022 - 2023

## Introduction

Ce projet avait pour objectif initial d'utiliser un programme de reconnaissance d'images basé sur le Deep Learning<sup>[1]</sup> pour identifier des mauvaises herbes. Pour atteindre cet objectif, nous avons suivi une approche progressive.

Dans un premier temps, nous avons choisi d'explorer un cas d'utilisation simple et courant en reconnaissance d'image, à savoir la reconnaissance de chiffres. Cette étape nous a permis de nous familiariser avec les concepts fondamentaux du Deep Learning, en particulier les réseaux de neurones convolutifs (CNN<sup>[2]</sup>). Nous avons étudié le fonctionnement et l'utilisation des CNN, en comprenant leur architecture et leurs différentes couches, ainsi que les techniques d'apprentissage et d'optimisation associées.

Une fois que nous avons acquis une compréhension solide des CNN, nous sommes passés à notre cas d'application spécifique : la reconnaissance de mauvaises herbes. Nous avons adapté nos connaissances et notre expérience acquises lors de la reconnaissance de chiffres pour résoudre ce problème particulier. Cela impliquait la collecte et la préparation d'un ensemble de données d'images de mauvaises herbes, la conception et l'entraînement d'un modèle CNN spécifique à ce domaine, ainsi que l'évaluation de ses performances.

# Reconnaissance de chiffres manuscrits

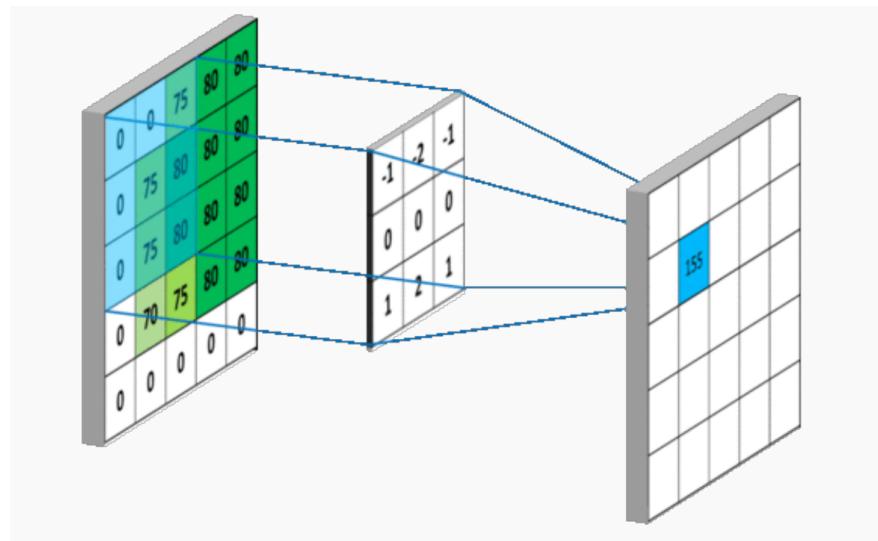
## Fonctionnement d'un CNN

Dans un premier temps, pour nous familiariser avec la reconnaissance d'image et le fonctionnement d'un CNN, nous avons développé un programme en C pour la reconnaissance de chiffres. Nous avons conçu notre propre CNN pour reconnaître des images de 25x35 pixels de chiffres, venant de notre base de données créée manuellement. Cette base est similaire à la base de données MNIST<sup>[3]</sup>, créée par le chercheur Yann Le Cun.

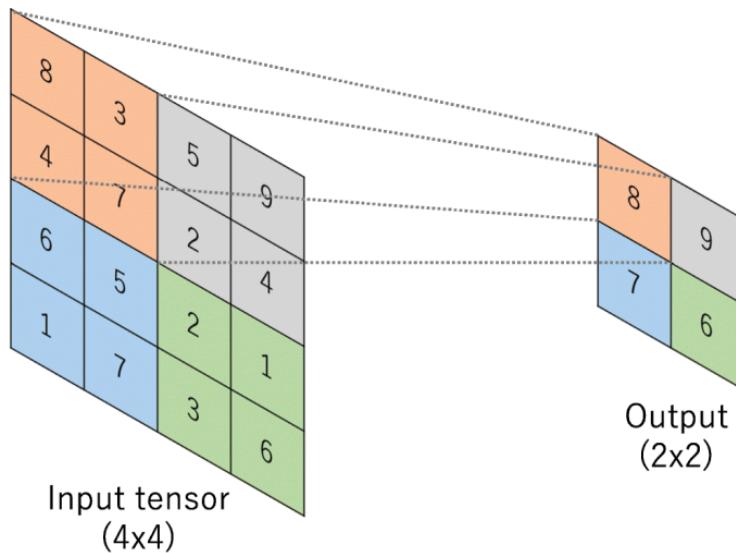
Cela nous a permis de découvrir le fonctionnement d'un CNN en détail. En effet, au lieu d'utiliser directement une bibliothèque de traitement d'images qui fonctionne comme une boîte noire que nous ne pouvons pas maîtriser et comprendre, nous nous sommes penchés en profondeur sur le fonctionnement d'un réseau de neurones. Ce dernier est basé sur un système de **poids**<sup>[4]</sup> (une matrice de coefficients actualisés à chaque itération). Le processus de formation d'un CNN implique l'ajustement des poids des filtres et des neurones pour minimiser la fonction de coût, qui mesure l'écart entre les prédictions du modèle et les étiquettes réelles de l'image d'entraînement. L'entraînement se fait par **rétropropagation du gradient**<sup>[5]</sup>, où les erreurs de prédiction sont propagées à travers le réseau et les poids sont ajustés en conséquence. Le modèle est évalué sur des données de validation pour déterminer quand il a atteint une bonne généralisation, c'est-à-dire lorsqu'il est capable de prédire avec précision des images qu'il n'a jamais vues auparavant.

Le fonctionnement d'un CNN peut être divisé en trois grandes étapes<sup>[1]</sup> :

**Convolution** : Dans cette étape, une série de filtres de convolution (également appelés noyaux) sont appliqués à l'image d'entrée. Chaque filtre est une petite matrice qui glisse sur l'image en effectuant des opérations de convolution. Cette opération de convolution calcule le produit scalaire entre les valeurs de chaque pixel de l'image et les valeurs de chaque pixel du filtre, puis elle somme tous les produits scalaires pour produire une nouvelle valeur pour chaque pixel de la sortie. Les filtres permettent de détecter des caractéristiques spécifiques de l'image, telles que les bords, les coins ou les textures. Pour des résultats optimaux, on utilise une base de filtres orthogonaux. Cela permet de s'assurer que chaque filtre reconnaît des caractéristiques différentes (un se concentre sur les variations verticales, un sur les variations horizontales, etc.). Nous utilisons dans notre cas une base orthogonale de 9 filtres de taille 3x3, générée par des calculs de transformée en cosinus.

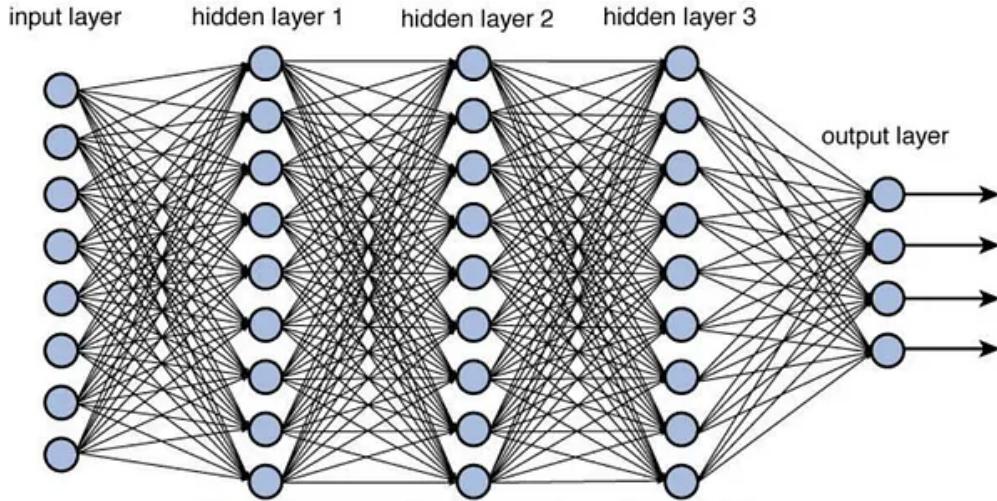


**Pooling :** Après chaque couche de convolution, une opération de pooling est effectuée. Le pooling permet de réduire la taille de l'image en prenant une zone de 2x2 pixels et en remplaçant cette zone par une seule valeur de pixel (dans notre cas, nous utilisons le *MaxPooling*, c'est-à-dire que nous prenons la valeur maximale des 4). Cette étape permet de réduire le nombre de paramètres et de rendre le réseau plus rapide et moins susceptible au sur-apprentissage.

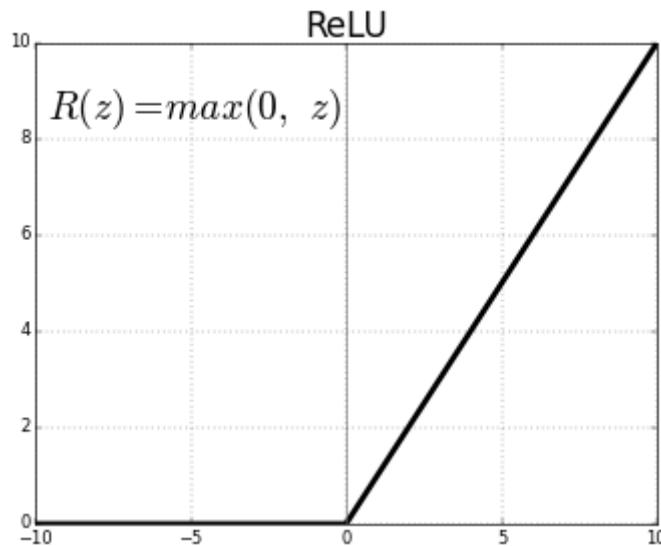


**Classification<sup>[2]</sup> :** Après plusieurs couches de convolution et de pooling, l'image est transformée en un petit nombre de caractéristiques qui sont ensuite utilisées pour la classification. Une ou plusieurs couches de neurones entièrement connectées (aussi appelées couches denses) sont utilisées pour convertir ces caractéristiques en probabilités pour chaque classe d'image. La couche d'entrée correspond aux valeurs des pixels de nos images, après être passés par plusieurs étapes de convolution et pooling. Ensuite notre réseau est composé d'un certain nombre de couches cachées (nous en utiliserons une seule, de taille correspondant

à la moyenne entre la taille de la couche d'entrée et la couche de sortie). La couche de sortie correspond au vecteur contenant les probabilités que notre image d'entrée appartienne à chacune de nos classes.



A ces étapes s'ajoute une fonction d'activation. La fonction d'activation ici utilisée (**ReLU** : fonction d'activation redresseur) est utilisée pour ajouter de la non-linéarité au modèle et pour permettre à celui-ci de modéliser des relations non-linéaires entre les entrées et les sorties.

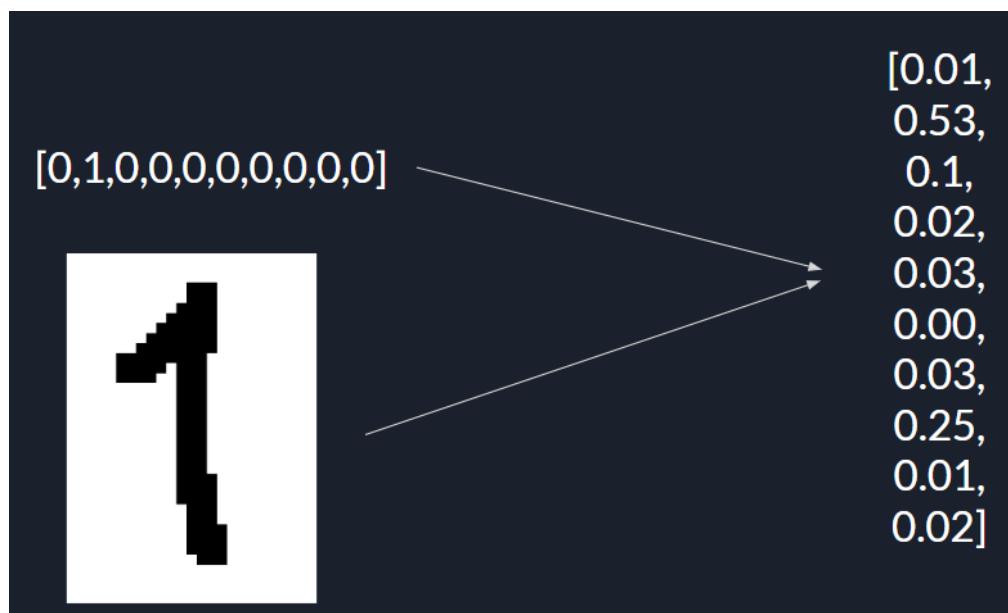


Au cours de l'algorithme, nous mémorisons et actualisons les poids associés à chaque lien entre les neurones de notre réseau. Ces poids correspondent à l'importance de la corrélation entre 2 neurones. Le poids le plus fort est donc attribué au lien le plus probable entre 2 neurones d'une couche à une autre. Ces poids sont générés aléatoirement puis actualisés à chaque itération de la phase d'**entraînement**, en calculant l'erreur entre le vecteur réponse souhaité (donné en

entrée avec l'image) et le vecteur obtenu. L'actualisation s'effectue par algorithme de rétropropagation du gradient, qui permet d'ajuster les poids en fonction de l'erreur commise.

Vient ensuite la phase de **validation**, qui permet d'évaluer la précision des prédictions de notre réseau en le testant avec un autre jeu de données (qu'il n'a donc jamais vu). Une fois l'erreur moyenne satisfaisante (inférieur à un seuil fixé), le réseau est finalisé et utilisable.

Vient enfin la phase de **test** qui permet de prédire la classe d'une image. En passant par les couches successives, le réseau détermine les probabilités que l'image en entrée appartienne à chacune des classes, et renvoie le vecteur de la couche de sortie. En entrée de notre algorithme, nous donnons notre image, ainsi que le vecteur souhaité en sortie (rempli de 0 sauf un 1 pour l'index du chiffre (par exemple : [0,1,0,0,0,0,0,0,0] si la figure est un 1). L'algorithme nous renvoie alors la prédition (sous forme de vecteur de probabilités) faite à l'aide du réseau précédemment généré.



Entrées et sorties de notre algorithme

Avant d'effectuer toutes ces étapes, nous devons effectuer un "preprocessing" qui permet d'adapter nos données (images brutes au format PNG) au format souhaité. Pour simplifier cette étape, nous collectons toutes les données à l'aide d'un programme Python pour les adapter à notre algorithme en C. Ce programme génère 2 fichiers CSV. Le premier correspond aux valeurs des pixels de toutes les images de la base de données, et le second aux vecteurs réponse souhaités. Ensuite, notre algorithme utilise ces 2 fichiers CSV pour générer le modèle et le tester avec différents chiffres.

## Reconnaissance de mauvaises herbes

### Présentation de l'environnement de travail

Après avoir travaillé sur la reconnaissance de chiffres, nous avons dû changer notre méthode pour l'adapter à notre cas d'étude. En effet, notre jeu de données de mauvaises herbes (basé sur le jeu de données [4Weed](#) [6], composé de 4 classes : cocklebur, foxtail, pigweed et ragweed) est beaucoup plus lourd que celui des chiffres. Avec des images en couleur de milliers de pixels, il est impossible d'utiliser le programme précédent, cela prendrait trop de temps et de puissance pour l'exécuter. C'est pourquoi nous avons décidé d'utiliser la bibliothèque Python [Keras](#) [7] de [TensorFlow](#). Nous avons divisé notre base de données de manière aléatoire en 3 dossiers : *train* (environ 80 % de la base), *validation* (environ 10 %) et *test* (environ 10 %). Chaque dossier est utilisé à une étape différente du processus.

Nom de l'espèce	Entraînement	Validation	Test	Total
Cocklebur	133	15	11	159
Foxtail	113	14	12	139
Pigweed	139	20	11	170
Ragweed	122	17	10	149
<b>Total</b>	<b>507</b>	<b>66</b>	<b>44</b>	<b>617</b>

Répartition de la base de données par espèce et par étape

## Présentation de notre modèle

Nous avons créé 2 programmes python, l'un qui permet de générer le modèle (avec le preprocessing, les différentes couches de convolution, et les phases d'entraînement et de validation), l'autre permet de l'exécuter pour prédire la classe d'une image (phase de test).

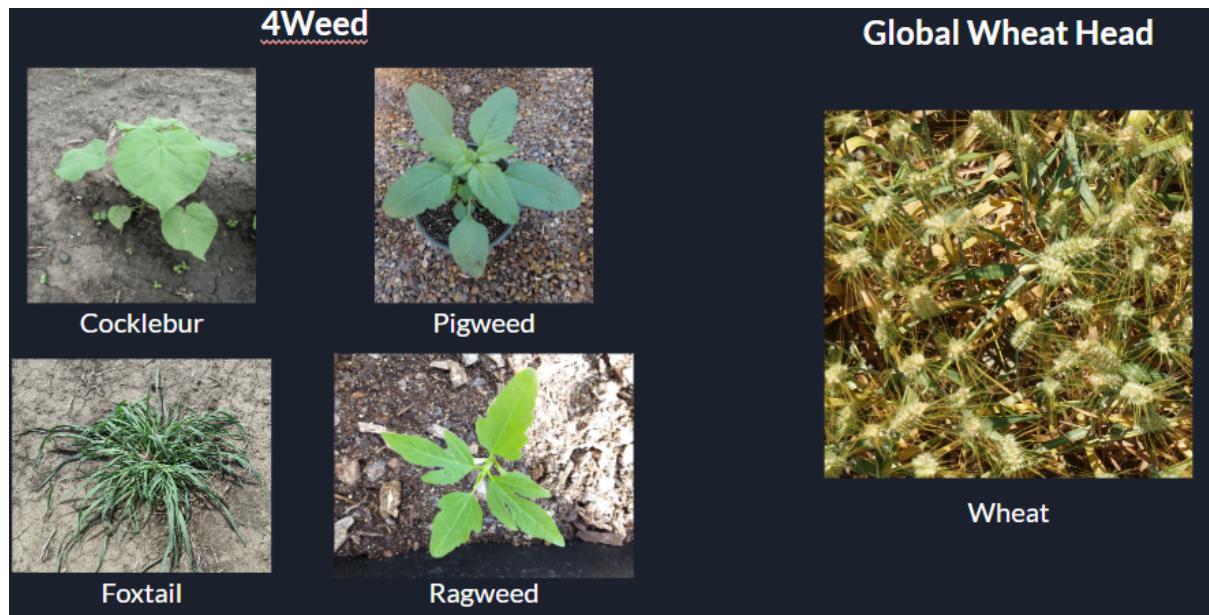
Dans notre modèle CNN, les noyaux de convolution sont de dimensions (3,3) et les filtres sont définis en utilisant le paramètre *activation='relu'*.

Nous utilisons 3 couches de convolutions. Le premier noyau de convolution est un filtre de taille 3x3 avec 16 canaux de sortie, le deuxième est un filtre de taille 3x3 avec 32 canaux de sortie, et le dernier est un filtre de taille 3x3 avec 64 canaux de sortie. Le nombre de canaux de sortie pour un noyau de convolution correspond au nombre de filtres de convolution qui sont appliqués à l'image en entrée.

Dans Keras, les filtres sont initialement choisis aléatoirement, puis ajustés au cours de l'entraînement pour améliorer la performance du modèle. Cela est donc assez peu pertinent puisque cela nécessite de la puissance de calcul inutile. En effet, comme dit précédemment, il est bien plus intéressant d'utiliser des filtres orthogonaux, pour être certain que chaque filtre mette en valeur des caractéristiques différentes. En utilisant des filtres générés aléatoirement, il est probable que plusieurs reconnaissent les mêmes caractéristiques, et soient donc inutiles.

Après avoir estimé les paramètres optimaux pour générer un CNN le plus précis possible, nous avons lancé notre génération avec un très grand nombre d'itérations (ou epoch) pour obtenir un modèle précis. Ce modèle est enregistré dans un fichier et nous l'utilisons ensuite avec un autre programme qui permet d'exploiter ce modèle pour effectuer des prédictions.

Notre objectif initial était de reconnaître des mauvaises herbes dans un champ de blé. Nous voulions donc ajouter à notre base de données une nouvelle classe "wheat", issue de la base de données *Global Wheat Head* [8]. Cependant nos 2 jeux de données ne sont pas compatibles car les mauvaises herbes sont isolées et ont un fond terreux, tandis que le blé recouvre toute l'image. Les résultats seraient donc les bons, mais seraient obtenus uniquement en analysant la couleur du fond, ce qui n'est pas pertinent. C'est pourquoi nous nous sommes concentrés sur la reconnaissance d'espèces de mauvaises herbes uniquement.



Incompatibilité des 2 bases de données

## Paramètres du modèle

Afin que la reconnaissance soit optimale, nous pouvons jouer sur plusieurs paramètres. Premièrement, nous pouvons jouer sur la taille de nos images et le nombre de pixels que l'on envoie dans notre algorithme. Afin d'avoir le maximum d'informations possible, nous avons pris la taille maximale qui nous était possible 512x512 sachant qu'à partir d'une certaine taille nos ordinateurs n'avaient pas la mémoire nécessaire de stocker toutes les valeurs pour effectuer les calculs si nous décidions de jouer sur les autres paramètres. Lors du prétraitement des images (preprocessing), les images sont donc redimensionnées pour atteindre cette taille. Il y a donc à cette étape une légère perte d'informations, qui n'aura presque pas de conséquences sur les résultats finaux. Plus on diminue la taille des images, plus la perte d'informations est importante, et plus les résultats finaux seront impactés.

Deuxièmement, nous pouvons jouer sur le nombre d'epoch c'est-à-dire le nombre de cycles d'apprentissage. Plus il y a d'epoch, plus l'apprentissage sera bon mais plus il prendra du temps. Nous avons choisi 150 epochs, ce qui est largement suffisant pour avoir des résultats pertinents, et qui demande déjà plusieurs heures de calculs.

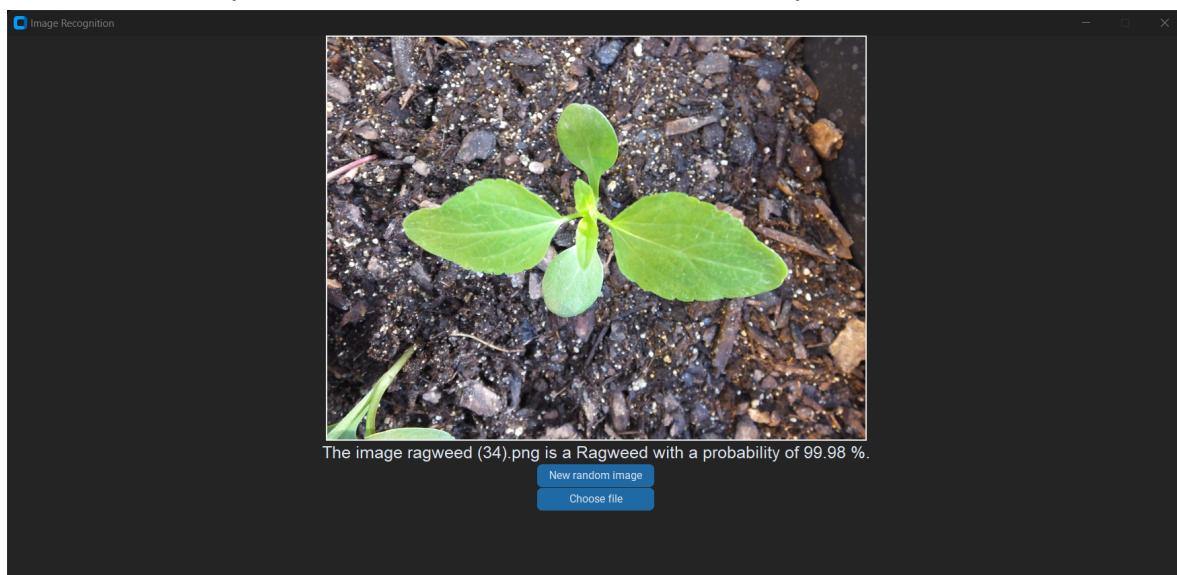
Enfin, nous pouvons régler le **batch\_size**<sup>[9]</sup> c'est-à-dire la taille d'un lot après lequel les poids sont mis à jour par rétropropagation du gradient. On a donc effectué plusieurs balayages de tailles de batch\_size afin d'observer comment l'apprentissage se déroule suivant les epoch choisies. On a pu constater que lorsque le batch\_size est très petit, la convergence de l'apprentissage est impossible, puisqu'à chaque étape le gradient est actualisé et change donc de direction. De même, lorsque le batch\_size est important par rapport à notre base de données (nombre et tailles des images), alors les poids ne sont pas actualisés assez régulièrement, ce qui rend la convergence très lente. Ainsi, après une série de tests, nous avons trouvé que le batch\_size optimal était de 45. Cette valeur n'est pas une valeur générale correspondant à n'importe quelle base de données. Elle est uniquement adaptée à notre banque d'images. Cependant, elle sera similaire pour des banques de mêmes ordres de grandeurs (dimensions et nombre d'images).

## Génération du modèle

Une fois avoir défini les paramètres les plus précis possibles, nous avons pu générer le modèle final. On réalise donc toutes les étapes détaillées précédemment pour calculer les poids du réseau. Les caractéristiques du modèle généré sont alors sauvegardées dans un fichier au format *h5* [\[10\]](#), un format de données hiérarchiques (HDF) utilisé pour stocker une grande quantité de données. Ce fichier est ensuite ouvert par notre programme qui permet l'exécution du modèle.

Dans un premier temps, nous avons développé un programme qui teste toutes les images de notre base de test. Cela nous a permis de comparer les résultats selon les paramètres choisis, et de les analyser.

Pour rendre notre travail plus utilisable et l'expérience utilisateur plus confortable, nous avons développé un second programme avec une interface graphique. Cette interface permet de réaliser une prédiction sur une seule image. Cette image peut être soit tirée aléatoirement parmi notre base de test (bouton “New random Image”) soit sélectionnée dans les fichiers de l'ordinateur (bouton “Choose file”). Une fois l'image choisie, elle est envoyée dans notre CNN, puis la photo et la prédiction correspondante s'affichent, comme dans la capture d'écran ci-dessous :



Capture d'écran de l'interface graphique

## Analyse des résultats

Lors de l'exécution du modèle, on donne en entrée du CNN générée chaque image de notre base de test. Le CNN renvoie alors une prédiction. Une prédiction est un vecteur qui contient les probabilités que l'image appartienne à chacune des classes. On s'intéresse donc à la probabilité la plus élevée. On considère la prédiction comme une réussite si la probabilité la plus élevée est bien celle de l'espèce à laquelle appartient l'herbe, et comme un échec sinon.

Lors de la phase de test, nous obtenons les résultats suivants :

Nom de l'espèce	Nombre de réussite	Moyenne des probabilités des réussites	Nombre d'échec	Moyenne des probabilités des échecs
Cocklebur	11	99,4%	0	
Foxtail	12	99,9%	0	
Pigweed	9	99,2%	2	70,0%
Ragweed	10	99,0%	0	
Total	42		2	

On constate que notre réseau fournit des résultats très satisfaisants par leur précision. En effet, on peut remarquer que les prédictions du modèle sont fausses pour seulement 2 images sur les 40 de notre base de test. En analysant les images concernées, nous réalisons qu'elles n'étaient pas vraiment exploitables (floue ou mal cadrée), et qu'il faudrait les supprimer de notre banque d'images, ou les traiter si possible :



Images à l'origine d'une erreur de prédiction

Pour vérifier la précision du modèle, nous l'avons également testé avec une dizaine d'images de chaque espèce trouvées sur internet. Les résultats obtenus sont aussi précis qu'avec les images de la banque 4Weed<sup>[6]</sup>, avec 15 réussites sur les 15 images choisies aléatoirement sur Google Image, et une moyenne de probabilité d'environ 99%. Ce test est pertinent puisqu'il permet de prendre du recul sur l'analyse de nos résultats. En effet, cela permet de s'assurer que les résultats ne sont pas biaisés par le fait que les images ont certainement été réalisées dans des conditions similaires (même appareil, même conditions météorologiques, même terre, etc.).

## Améliorations possibles

Bien que les résultats obtenus soient plutôt convaincants, plusieurs axes d'amélioration pourraient être envisagés pour rendre ce projet plus complet.

Premièrement, notre base de données est suffisamment grande pour avoir de bons résultats, mais devrait être bien plus conséquente pour pouvoir les exploiter de manière pertinente. En effet, quelques centaines d'images ne sont pas suffisantes pour obtenir de bons résultats sous n'importe quelle condition (exemple de l'image floue ou mal cadrée cité précédemment). Une solution à ce problème pourrait être la **data augmentation**<sup>[11]</sup>, qui consiste à modifier légèrement les données (par translation ou rotation par exemple) pour en former de nouvelles. Cette technique permet d'élargir facilement une base de données, ce qui réduit le risque de surentrainement.

Par ailleurs, l'objectif final de ce projet n'était pas uniquement la reconnaissance d'images de mauvaises herbes, mais d'en faire une réelle intégration qui puisse être utilisable en pratique. Plusieurs possibilités sont envisageables, mais nous pensions dans un premier temps rendre possible la reconnaissance de mauvaises herbes à partir d'un flux vidéo, et non uniquement des photos. Cela permettrait d'identifier les zones de mauvaises herbes dans une vaste étendue (ex : champ de blé). Une fois cette fonctionnalité ajoutée, il sera alors possible de l'intégrer dans un système embarqué, telle qu'une Raspberry Pi connectée à un drone avec une caméra.

## Conclusion

Ce projet a été une occasion précieuse pour développer nos compétences et acquérir des connaissances approfondies en Deep Learning, en particulier sur les réseaux de neurones convolutifs (CNN). En explorant en détail le fonctionnement des CNN, nous avons pu prendre du recul sur l'utilisation de bibliothèques de reconnaissance d'images telles que Keras, ainsi que sur l'importance et la pertinence des bases de données utilisées dans le processus.

Cette expérience nous a également ouvert des perspectives intéressantes pour des applications futures. Par exemple, ce projet pourrait être adapté à une utilisation embarquée, où des caméras pourraient être utilisées pour déterminer les zones nécessitant l'application de produits de traitement. Cela pourrait être particulièrement utile dans des domaines tels que l'agriculture ou l'entretien des espaces verts, où la détection des mauvaises herbes est cruciale pour optimiser l'utilisation des produits de traitement et réduire les coûts environnementaux.

## Références

[1] *Deep Learning* - I. Goodfellow, Y. Bengio et A. Courville, MIT Press, 2016 :  
<https://www.deeplearningbook.org/>

[2] *Gradient-based learning applied to document recognition* - Y. LeCun, L. Bottou, Y. Bengio et P. Haffner, Proceedings of the IEEE, 1998 :  
[http://vision.stanford.edu/cs598\\_spring07/papers/Lecun98.pdf](http://vision.stanford.edu/cs598_spring07/papers/Lecun98.pdf)

[3] Base de données MNIST, par Yann Le Cun : <https://yann.lecun.com/exdb/mnist/>

[4] Calcul des poids lors de la rétropropagation du gradient :  
<http://www.anyflo.com/bret/cours/rn/rn5.htm#retropropagation>

[5] La rétropropagation du gradient :  
<https://www.miximum.fr/blog/introduction-au-deep-learning-2/>

[6] Base de données 4Weed, pour les mauvaises herbes : <https://arxiv.org/abs/2204.00080>  
Lien de téléchargement : <https://osf.io/w9v3j/>

[7] Bibliothèque Keras de TensorFlow : <https://www.tensorflow.org/guide/keras?hl=fr>

[8] Base de données Global Wheat Head, pour le blé :  
<https://zenodo.org/record/5092309#.ZD6L7nZBy3B>

[9] Explication de la Batch Size :  
<https://medium.com/geekculture/how-does-batch-size-impact-your-model-learning-2dd34d9fb1fa>

[10] Format de fichier h5 : <https://docs.fileformat.com/fr/misc/h5/>

[11] *ImageNet Classification with Deep Convolutional Neural Networks* - A. Krizhevsky, I. Sutskever et G. E. Hinton, Advances in Neural Information Processing Systems (NIPS), 2012 :  
[https://papers.nips.cc/paper\\_files/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html](https://papers.nips.cc/paper_files/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html)

Publications scientifiques et projets liés à la reconnaissance de mauvaises herbes :

- *Détection automatique de plantes au sein d'images aériennes de champs par apprentissage non supervisé et approche multi-agents* - E. Jacopin :

<https://hal.science/hal-03650231>

- *Systèmes de vision numérique appliqués à la gestion d'intrants en agriculture : vers le contrôle de la pulvérisation et de l'épandage* - S. Vilette : <https://hal.science/hal-01840843>

- Projet AMAZONE UX SmartSprayer :

<https://amazone.fr/fr-fr/agritechnica/nos-nouveaut%C3%A9s-agritechnica-2022/nouveaut%C3%A9s-d%C3%A9tail/ux-smartsprayer-amazone-998762>

- Projet Ecorobotix : <https://ecorobotix.com/fr/>

GitHub de ce projet : <https://github.com/HugoC28/ImageRecognition>