

## Rapport de stage

Laboratoire d'Informatique en Calcul Intensif et Image  
pour la Simulation (LICIIS)

-

BUDAI Noah

Du 07/04/2025 au 30/05/2025

Tuteur de stage : Lilian HOLLARD

Tuteur enseignant : M. Cyril RABAT

Établissement : URCA - UFR SEN

Diplôme : Licence 2 Informatique - 2024/2025



## Remerciements

Je tiens à exprimer ma gratitude aux personnes qui ont contribué au bon déroulement du stage.

Je remercie particulièrement HOLLARD Lilian, mon encadrant de stage, pour ses précieux conseils et sa disponibilité tout au long du stage, sa pédagogie m'a aidé à acquérir de solides bases en IA.

Je remercie également le LICIIS, rattaché à la faculté des Sciences Exactes et Naturelles de l'Université de Reims Champagne-Ardenne de m'avoir chaleureusement accueilli et en particulier STEFFENEL Luiz-Angelo.

# Table des matières

I.	Présentation de l'entreprise .....	4
1)	La structure .....	4
2)	Le contexte .....	4
II.	Description du stage .....	5
1)	Auto-formation Machine Learning et Deep Learning.....	5
2)	Création de mes propres modèles convolutionnels.....	10
III.	Présentation de mes modèles convolutionnels .....	14
a)	Modèle simple .....	14
b)	Modèle très profond.....	15
c)	Petit modèle résiduel : utilité de la data augmentation .....	15
d)	Modèle résiduel : stride = 2 comparé à MaxPool2d .....	17
e)	Optimisation du modèle résiduel.....	18
IV.	Conclusion .....	19
V.	Annexes .....	20

## I. Présentation de l'entreprise

### 1) La structure

Mon stage a été effectué au sein du Laboratoire d'Informatique en Calcul Intensif et Image pour la Simulation (LICIIS), lui-même rattaché à la Faculté de Sciences Exactes et Naturelles de l'Université de Reims Champagne-Ardenne.

Ce laboratoire effectue des recherches sur des thématiques variées, telles que :

- Les performances et les modèles de programmation d'architectures hybrides comportant des accélérateurs de calcul et les applications du calcul scientifique pouvant en tirer parti ;
- La mise en œuvre et l'exploitation de clusters hybrides ;
- La visualisation scientifique en environnement HPC.

Ces thèmes impliquent également une recherche dans l'Intelligence Artificielle (IA), domaine où j'ai eu l'opportunité de mener un travail approfondi et méthodique durant ce stage.

### 2) Le contexte

J'ai été encadré par Monsieur HOLLARD Lilian, doctorant en dernière année, qui m'a accompagné et m'a fixé différents objectifs à atteindre tout au long du stage.

Certains doctorants du LICIIS mènent des travaux sur l'application du Deep Learning à des images de plantes malades. Dans le même objectif, mon stage avait pour but de développer mon propre modèle de classification d'images, capable de détecter un certain nombre de maladies sur différentes feuilles. Pour cela, j'ai travaillé sur les réseaux de neurones en utilisant PyTorch, le framework plus recherché sur le marché du travail.

On m'a également conseillé le jeu de données PlantVillage, contenant environ 55 000 images bien étiquetés de manière à pouvoir se concentrer sur l'amélioration du modèle et non pas le format des images et leur exploitabilité.

L'objectif principal était d'atteindre une précision élevée dans la détection automatique de maladies de plantes à partir d'images de feuilles. Pour cela, j'ai d'abord approfondi mes connaissances en intelligence artificielle de manière autonome, puis j'ai été guidé afin de concevoir un modèle performant de manière indépendante. Ce stage fut donc réalisé la majeure partie du temps en distanciel, avec quelques séances en présentiel.

## II. Description du stage

### 1) Auto-formation Machine Learning et Deep Learning

Pour commencer ce stage avec de solides fondations, il a fallu dans un premier temps que je me forme en autonomie, à l'aide de diverses ressources récupérées sur Internet à propos du Machine Learning et du Deep Learning. Cette phase d'auto-formation s'est appuyée sur des tutoriels / cours en ligne (voir annexes). Je vais, ici, citer les deux ressources qui m'ont apporté le plus de connaissances.

Premièrement, le livre « Apprendre le Machine Learning en une semaine » de Guillaume SAINT-CIRGUE, qui m'a introduit au monde de l'IA avec les concepts fondamentaux qui m'ont servi tout au long du stage. J'y ai découvert l'apprentissage supervisé, notamment ses deux grandes branches : la régression et les classifications. C'est l'apprentissage le plus populaire et le plus utile en Deep Learning, j'y ai donc porté une attention particulière.

Figure 2 : Exemple de régression linéaire  
Prix / Surface

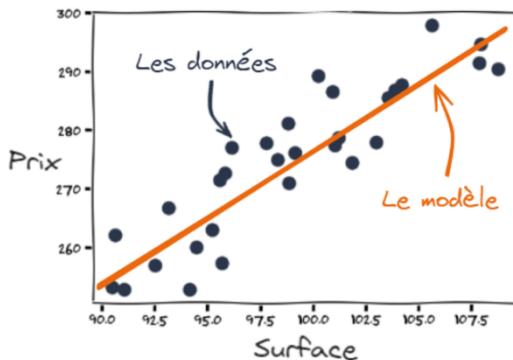
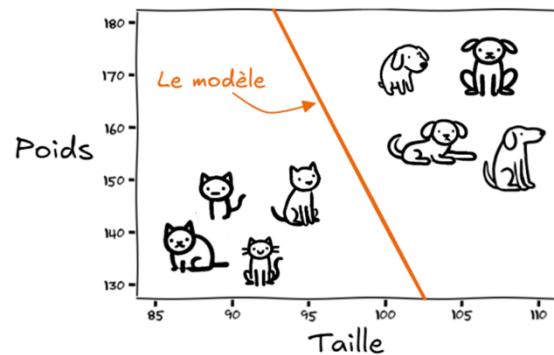


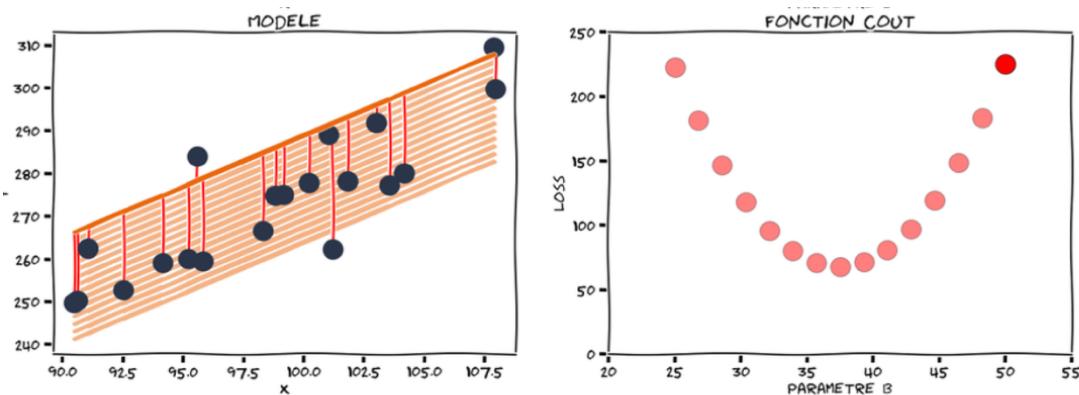
Figure 1 : Exemple de classification  
Chat / Chien



Source : Apprendre le Machine Learning en une semaine », Guillaume SAINT-CIRGUE

Pour commencer, concernant la **régression linéaire**, un algorithme marquant était la **descente de gradient**. Ce dernier permet d'ajuster les coefficients du modèle afin de minimiser la fonction de coût (l'erreur) pas à pas. Il est essentiel à comprendre et m'a fait également revoir des concepts mathématiques comme les dérivées partielles et les fonctions composées. Cette **fonction de coût** (Loss function) est aussi un concept essentiel ; c'est un point de repère entre la sortie du modèle et la sortie attendue. On calcule pour chacun des points du jeu de données cette « loss » avec différents paramètres, ici  $b$  :

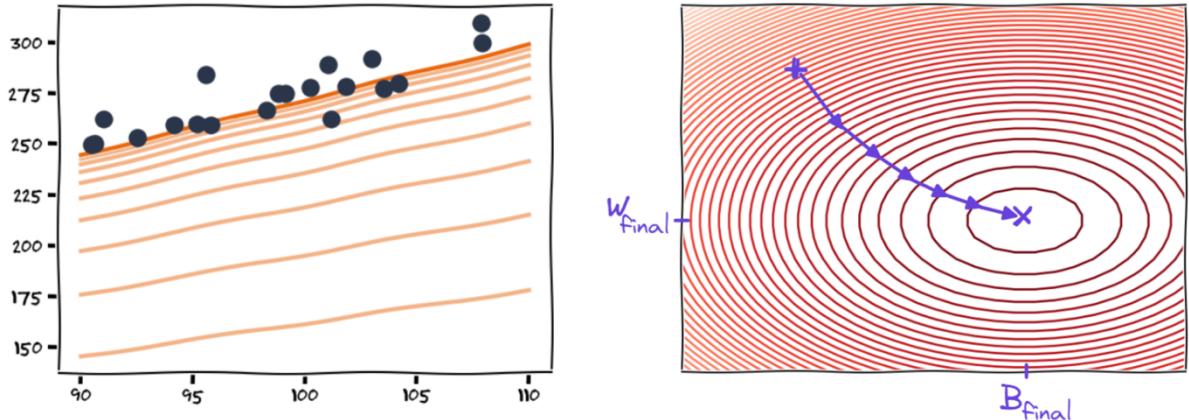
Figure 3 : Variation de la fonction coût en fonction du paramètre  $b$



Source : Apprendre le Machine Learning en une semaine », Guillaume SAINT-CIRGUE

Ces notions m'ont permis de comprendre que l'objectif principal dans la création d'un modèle est de minimiser la fonction de coût, en ajustant les paramètres du modèle via la [desccente de gradient](#). Cette dernière avec les deux paramètres  $w$  et  $b$  (de la régression  $w \cdot x + b$ ) ressemble à ceci, on converge petit à petit vers le point où l'erreur est minimale :

Figure 4 : Descente de gradient et ajustement des paramètres

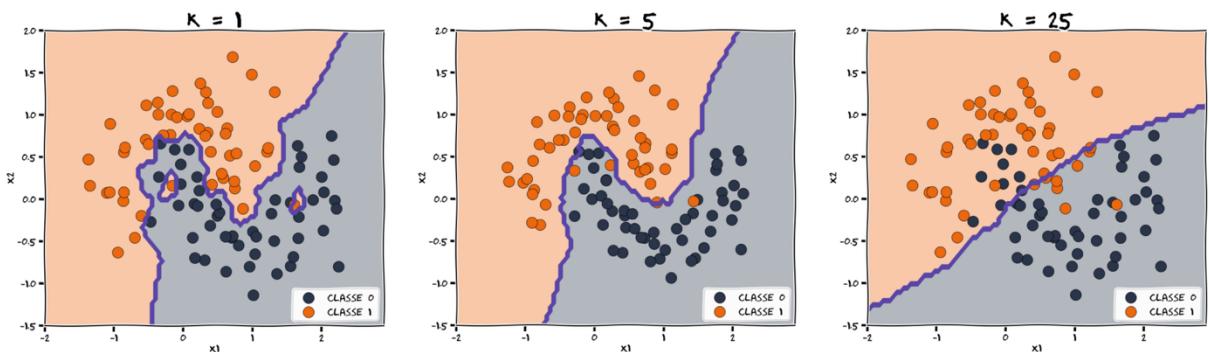


Source : Apprendre le Machine Learning en une semaine », Guillaume SAINT-CIRGUE

Les mathématiques sous-jacentes impliquent pour chaque mise à jour d'un paramètre de récupérer la valeur du paramètre actuel et de lui soustraire le gradient (la dérivée partielle de la fonction de coût par rapport au paramètre actuel) jusqu'à convergence.

Pour la **classification**, j'ai pu analyser et intégrer quelques algorithmes comme le Nearest Neighbors. Il s'agit simplement d'attribuer à un nouveau point la même classe que le point le plus proche de lui, ce qui amène à former une « frontière de décision ». En approfondissant, on apprend que l'algorithme du [K-Nearest Neighbors](#) est beaucoup plus efficace ; on regarde un nombre de  $k$  voisins les plus proches et on assigne la classe la plus représentée parmi les  $k$  voisins. Cela permet d'éviter des erreurs de classifications avec certains cas qui seraient aberrants.

Figure 5 : K-Nearest Neighbors avec des valeurs de  $K$  différentes



Source : Apprendre le Machine Learning en une semaine », Guillaume SAINT-CIRGUE

Je me suis également renseigné sur l'**apprentissage non supervisé**. Je suis passé par le clustering avec l'algorithme du K-Means Clustering où le principe est d'avoir la machine qui forme ses propres clusters (on lui donne l'entrée et elle détermine la sortie). J'ai également vu certaines méthodes de détections d'anomalies comme le Local Outlier Factor, et des méthodes d'optimisation comme la méthode du coude permettant de minimiser la variance sans tomber dans la suroptimisation.

J'ai ensuite été particulièrement intrigué à l'idée de la création d'un agent libre lorsque j'ai découvert **l'apprentissage par renforcement**. Celui-ci peut entreprendre des actions et se perfectionne selon les expériences vécues. J'ai donc appris le [Q-Learning](#); la machine explore son environnement et génère des données qui forme une fonction Q. Cette dernière permet d'indiquer la récompense obtenue en choisissant une certaine action dans un certain état. Pour aller plus loin, on peut se servir de l'équation de Bellman, qui a pour concept de prendre en compte les possibles récompenses futures dans le choix présent.

Figure 6 : Tableau représentant la fonction Q

$S \setminus A$	$\uparrow$	$\rightarrow$	$\downarrow$	$\leftarrow$
$S$	0.34	-0.18	1.69	-0.45
$A$	0.81	-0.74	0.98	0.49
$S \setminus A$	-0.63	-0.90	0.29	0.73
$A$	⋮	⋮	⋮	⋮

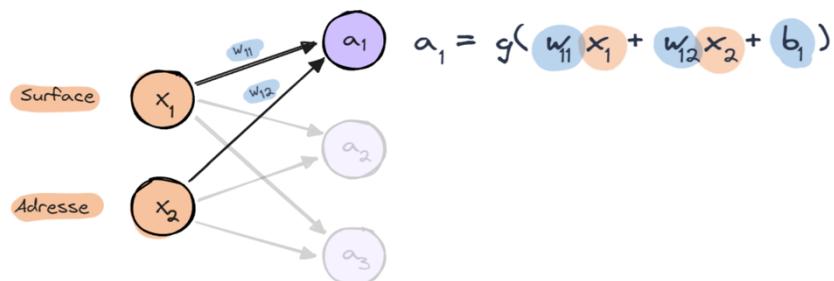
- ⇒ S = State, c'est-à-dire l'état actuel de l'agent
- ⇒ A = Action
- ⇒ Q(S,A) = Récompense associée

Source : Apprendre le Machine Learning en une semaine », Guillaume SAINT-CIRGUE

Finalement, ce livre comporte une légère introduction aux **réseaux de neurones**, partie la plus pertinente pour la future construction de mon modèle. J'ai donc appris ce qu'était un réseau de neurones ainsi que leur utilité. Ils sont particulièrement intéressants par le fait d'avoir plusieurs couches de neurones réalisant plusieurs combinaisons linéaires de chaque entrée pour produire une unique sortie.

On introduit également une **fonction d'activation** sur chaque sortie, ce qui permet de modéliser des fonctions beaucoup plus complexes.

Figure 7 : Fonction d'activation g sur l'ensemble des combinaisons linéaires



Source : Apprendre le Machine Learning en une semaine », Guillaume SAINT-CIRGUE

Grâce à cette partie du livre, j'en sais plus sur les grandes architectures de réseaux de neurones ; comme les RNN (Recurrent Neural Network) utile pour l'analyse de texte, les Auto-Encodeurs pour les DeepFake, les transformers pour les LLM (Large Language Model) comme ChatGPT et le plus pertinent : les **CNN** (Convolutional Neural Network) pour le traitement d'image et la détection d'objets.

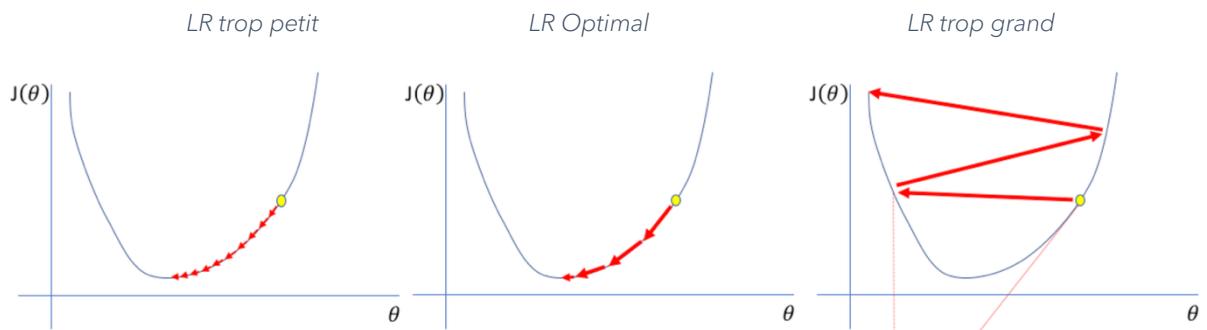
Ainsi, je suis passé à ma seconde ressource : « learnpytorch.io ». Elle m'a permis de faire le lien entre ce que j'avais déjà appris et le code associé. J'ai également pu approfondir considérablement mes connaissances en Deep Learning. De plus, la ressource est en anglais ce qui m'a permis de remobiliser des compétences linguistiques.

J'ai commencé par découvrir les **tenseurs** ; des tableaux de n-dimensions de nombres qui vont permettre de représenter des données. J'ai appris à les manipuler, les indexer et les redimensionner. C'est un concept majeur en Deep Learning car ils peuvent représenter presque tout ce que l'on souhaite, il est donc important de savoir les utiliser. Ces tenseurs représentent souvent des matrices, je me suis donc exercé sur la multiplication matricielle et leurs transposés.

J'ai donc fait mes premiers pas avec le framework PyTorch. En me renseignant, j'ai découvert l'importance d'exécuter les calculs sur la carte graphique plutôt que sur le processeur, car cela est plus optimisé grâce au parallélisme. Je me suis ensuite concentré sur le fait de construire tout l'environnement de travail nécessaire :

- D'abord, j'ai appris à récupérer ou transformer les données correctement afin de pouvoir les manipuler sous forme de tenseurs et les séparer en **3 sets** : un set d'entraînement, un set de validation et un set de test. Le modèle apprendra sur le set d'entraînement, se testera sur celui de validation (un peu comme des « révisions » avant le test) et finalement on pourra l'évaluer sur le set de test.
- Puis, mis en place un premier modèle très simple, une régression linéaire pour prendre en main les différents modules de PyTorch. Ce modèle est une classe héritant d'autres modules PyTorch (`nn.Module`). Il contient notamment une fonction (`forward`) qui va définir les opérations qu'on appliquera sur nos données passées en paramètre (ici,  $ax+b$  avec  $a$  et  $b$  les paramètres à définir et améliorer).
- Par la suite, revu la notion de fonction de coût comme précédemment expliquée et aussi celle d'**optimizer**. Cette notion d'optimizer qui m'a paru nouvelle au départ n'est en fait qu'un outil qui permet de mettre à jour les paramètres du modèle en fonction de l'erreur calculée. Il existe beaucoup d'optimizer, j'ai donc étudié les choix à ma disposition sur la documentation officielle et ai choisi le plus simple pour commencer (Stochastic Gradient Descent, une version améliorée de la descente de gradient). De cette manière, j'ai également découvert **le taux d'apprentissage** (Learning Rate = LR) qui est la « longueur » du pas que l'on veut prendre dans la direction du gradient, il est choisi par expérimentation.

Figure 8 : différents taux d'apprentissage et leur impact sur la découverte du minimum



Source : « <https://www.jeremyjordan.me/nn-learning-rate/>

- Ensuite, j'ai construit une **boucle d'entraînement** du modèle. On commence par une phase d'entraînement : on passe les données d'entraînement au modèle, on calcule ensuite l'erreur puis on fait la **back-propagation** (dérivation en chaîne pour comprendre comment ajuster chaque paramètre) et finalement on applique notre optimiseur donc ici on fait la descente de gradient (mettre à jour les poids pour diminuer l'erreur). A chaque fois, on fait également une phase de validation en passant les données de validation et en calculant l'erreur pour voir si le modèle généralise bien sur de nouvelles données.
- Finalement, j'ai appris à faire des **prédictions** avec le modèle entraîné sur le jeu de données FashionMNIST. Pour suivre les performances j'ai également instauré une métrique de **précision** du modèle et une **matrice de confusion**. Cette matrice permet de visualiser là où il a le plus de mal à faire ses prédictions et avec quel élément il peut confondre.

Figure 9 : Différences entre prédition et vrai label



Figure 10 : Matrice de confusion

	T-shirt/top	Trouser	Pullover	Dress	Coat	Sandal	Shirt	Sneaker	Bag	Ankle boot
true label	852	2	11	30	3	1	94	0	7	0
T-shirt/top		972	0	15	3	0	5	0	1	0
Trouser	4		748	11	129	0	97	0	2	0
Pullover	12	1		913	20	0	30	0	2	0
Dress	22	4	9		824	0	86	0	0	0
Coat	2	2	46	40		0	666	0	13	0
Sandal	1	0	0	2	0	977	0	14	1	5
Shirt	150	0	55	33	83	0		946	0	32
Sneaker	0	0	0	0	0	22	0		957	0
Bag	3	1	3	8	3	4	16	5		950
Ankle boot	0	0	0	1	0	10	1	37	1	
predicted label	T-shirttop	Trouser	Pullover	Dress	Coat	Sandal	Shirt	Sneaker	Bag	Ankle boot

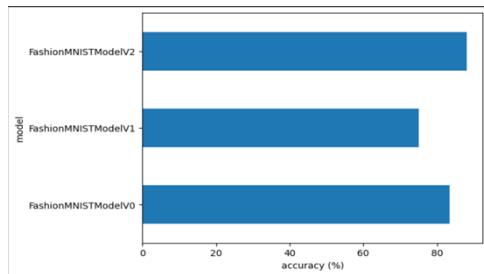
Source : production personnelle

- Afin de pouvoir réutiliser le modèle ultérieurement j'ai découvert comment le **sauvegarder** et le charger au besoin.

Pendant toutes ces expérimentations, j'ai beaucoup visualisé à l'aide notamment de graphiques matplotlib ou même de DataFrame de la bibliothèque pandas, ces notions vues en cours m'ont été très utile. J'ai par exemple pu voir [l'évolution de la précision](#) de mes différents modèles (voir ci-dessous)

Figure 11 : Précision des différents modèles construits sur le jeu de données FashionMNIST

Source : production personnelle

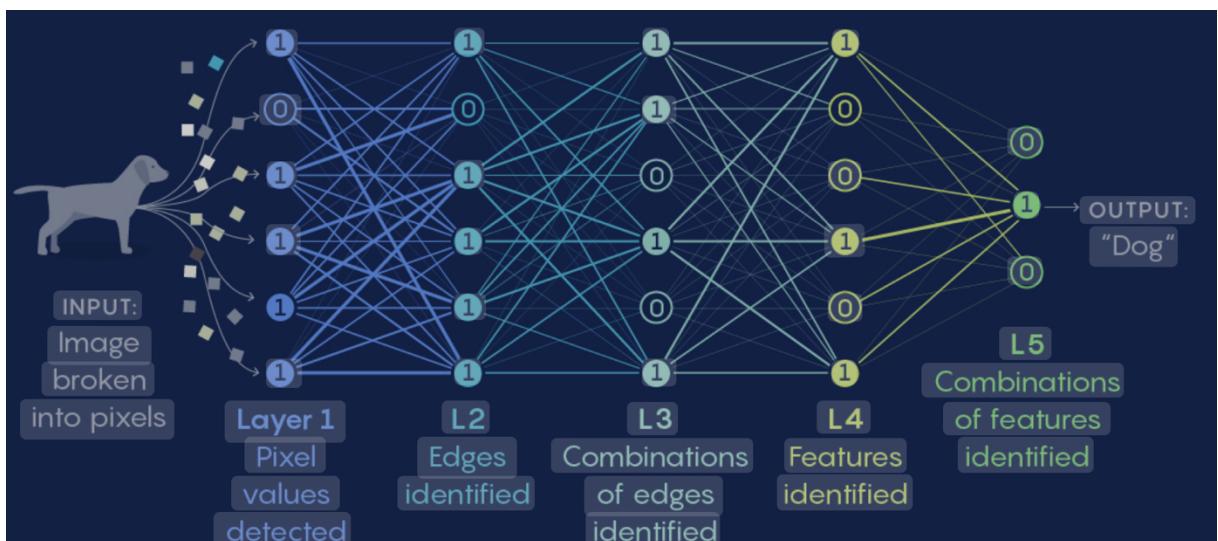


## 2) Création de mes propres modèles convolutionnels

Une fois de solides bases fondées, mon encadrant de stage m'a formé sur les architectures et les concepts les plus communs dans les réseaux de neurones, en particulier les **CNN** (Convolutionnal Neural Network). On a d'abord fait un point sur les fonctions d'activations et leurs aspects, par exemple pour **ReLU** (Rectified Linear Unit) ou Sigmoid ainsi que leur utilité dans la réalisation de fonctions complexes par extension de motifs plus complexes. J'ai donc pu aborder mon premier modèle de classification, au départ juste un choix binaire avec l'utilisation de nouvelles fonctions de coût (Binary Cross Entropy Loss), puis des classifications multiples (Cross Entropy Loss) où j'ai pu visualiser les avantages de la **non-linéarité** et également revoir les **probabilités** (pour récupérer la classe prédite par la fonction softmax).

Une fois ce nouveau bagage de connaissances acquis, l'objectif était de préparer mon premier modèle **convolutionnel**. Le but est de « donner la vue » à l'ordinateur, pour qu'il reconnaisse des motifs et des couleurs. Or, les jeux de données contenant les images sont souvent plutôt lourds (2Go), ce qui m'a amené à étudier les **DataLoader**, qui sont en fait des itérateurs permettant de charger les images lot par lot. Le point essentiel est la convolution, qu'on pourrait expliquer par le fait de reconnaître des patterns dans des données visuelles. J'ai étudié ce concept de **couche convolutionnelle** qui consiste à glisser un filtre le long de l'image en extrayant des formes de plus en plus complexes.

Figure 12 : Schématisation du fonctionnement des couches convolutionnelles



Source : « <https://fr.pinterest.com/pin/822892163184479166> »

Par la suite j'ai notamment expérimenté différentes tailles de filtres, différentes marges (padding), plus ou moins de couches convolutionnelles... Puis, j'ai appris le concept du **MaxPooling**, le fait de « compresser » l'image en gardant seulement l'information la plus forte sur un filtre d'une certaine taille qui parcourt l'image.

Figure 13 : Concept du Max Pooling

12	20	30	0
8	12	2	0
34	70	37	4
112	100	25	12

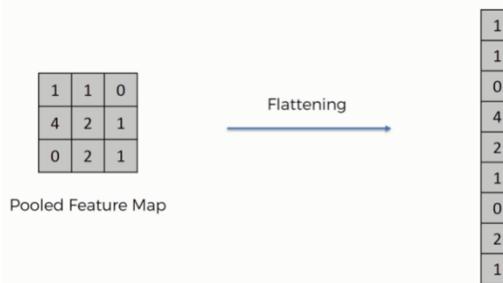
$2 \times 2$  Max-Pool

20	30
112	37

Source : « <https://paperswithcode.com/method/max-pooling> »

Le but étant à terme de passer d'une image 3D à des prédictions de classes, j'ai dû m'intéresser à la fonction **Flatten** qui permet « d'aplatir » les sorties des différentes couches en un vecteur 2D, ainsi qu'à la fonction **Linear**, pour transformer le vecteur en prédictions.

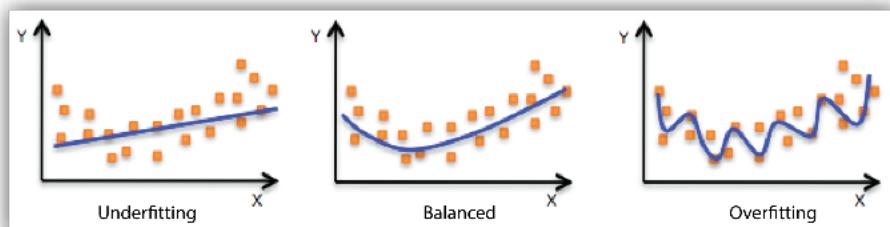
Figure 14 : Illustration de la fonction Flatten



Source : « <https://www.superdatascience.com/blogs/convolutional-neural-networks-cnn-step-3-flattening> »

C'est également lors d'expérimentations et différents entraînements avec ce modèle que j'ai commencé à découvrir le phénomène **d'overfitting**, c'est-à-dire lorsque le modèle apprend juste les exemples sans les généraliser. Je me suis donc renseigné sur différentes méthodes pour réduire ce phénomène : réduire la complexité (enlever des couches de neurones), ajuster le taux d'apprentissage... En étudiant ces possibilités, j'ai choisi de rajouté dans mon modèle du **BatchNorm**, permettant de normaliser les activations d'une couche pour chaque lot en centrant les valeurs sur 0 avec un écart-type de 1.

Figure 15 : Différents phénomènes lors de l'entraînement du modèle



Source : [https://docs.aws.amazon.com/fr\\_fr/machine-learning/latest/dg/model-fit-underfitting-vs-overfitting.html](https://docs.aws.amazon.com/fr_fr/machine-learning/latest/dg/model-fit-underfitting-vs-overfitting.html)

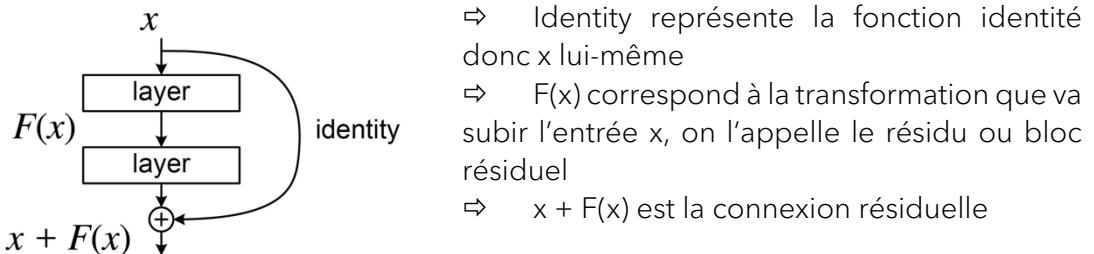
Finalement, j'ai pu expérimenter et approfondir mes connaissances en créant de multiples modèles dont je devais constamment améliorer les performances. Au départ, après avoir chargé le jeu de données PlantVillage et entraîner mon premier modèle, il n'arrivait pas à apprendre : toutes les prédictions étaient aléatoires. C'est ainsi que j'ai expérimenté avec plein d'autres modèles, certains avec plus de couches, d'autres avec ou sans max pooling, certains avec moins de couches, avec plus de fonctions non linéaires...

J'ai dû faire des recherches qui m'ont amené vers la « **data augmentation** » qui consiste à créer artificiellement plus d'images pour que le modèle généralise mieux. Dans la même logique, j'ai appris le **Dropout** qui consiste à désactiver aléatoirement certains neurones pendant l'entraînement pour éviter la dépendance d'un petit groupe seulement. Il y a beaucoup d'autres concepts que j'ai eu la possibilité d'essayer comme le early stopping et le gradient clipping.

Cependant, une plus grande problématique empêchait mon modèle d'apprendre correctement : le **vanishing gradient**. En effet, avec beaucoup de couches, les dérivées deviennent de plus en plus petites, dû à la dérivation en chaîne on finit par multiplier des valeurs inférieures à 1 et le gradient devient presque nul, donc il n'y a pas d'apprentissage.

La solution pour pallier ce problème est d'utiliser des **connexions résiduelles**. L'objectif est d'additionner l'entrée avec la sortie du bloc dans le but de la transmettre inchangée si le réseau ne peut pas améliorer l'entrée. J'ai directement pu observer des résultats très satisfaisants, en passant de prédictions aléatoires (environ 10% de précision) à des prédictions quasiment exactes (environ 95%).

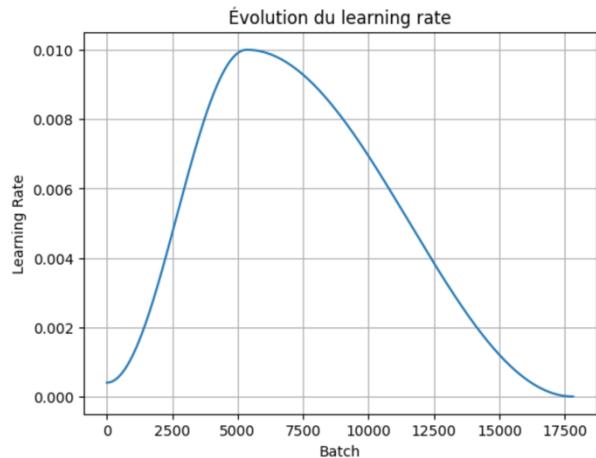
Figure 16 : Bloc Résiduel



Source : « [https://en.wikipedia.org/wiki/Residual\\_neural\\_network](https://en.wikipedia.org/wiki/Residual_neural_network) »

Par la suite, j'ai beaucoup expérimenté pour obtenir plus de précision. Le passage à 96% s'est fait grâce au remplacement du Max Pooling par un stride (un pas) de 2 lors de la convolution. Ensuite, j'ai gagné un peu de précision en ajoutant plus de transformations lors de la data augmentation tel que des rotations aléatoires sur les images. Ce qui m'a vraiment permis d'obtenir la dernière augmentation significative de précision fut de rajouter un Learning Rate (LR) Scheduler et du weight decay. Le **LR Scheduler** permet d'ajuster dynamiquement le taux d'apprentissage lors de l'entraînement pour éviter qu'il soit bloqué dans un minimum local ou qu'il soit trop lent.

Figure 17 : Évolution du LR au cours de l'entraînement



➤ On observe bien l'ajustement dynamique du LR, des pas très grand au départ puis qui diminuent au fur et à mesure

Source : Production personnelle

Concernant le **weight decay**, il pénalise les poids trop grands pour améliorer la généralisation. C'est en associant l'ensemble des notions vues précédemment que le modèle a atteint une précision de 99%.

Pour finir, j'ai cherché à **optimiser** mon modèle, en tentant de réduire son nombre de paramètres sans pour autant qu'il perde en précision. Ceci est très utile surtout pour réduire le temps d'entraînement et donc toute la consommation énergétique que cela implique. Encore une fois, après diverses expérimentations, j'ai réduit les paramètres en passant de 5 millions à 1 million, avec la même précision et plus de 10 minutes gagnés sur l'entraînement. Pour cela, j'ai simplement réorganisé les couches et en ai supprimé quelques-unes.

### III. Présentation de mes modèles convolutionnels

J'ai décrit l'évolution de mes modèles tout au long de ce rapport, voici alors mes principaux modèles qui ont marqués mes avancés (seulement ceux comportant de la convolution). Tous les entraînements des modèles se sont fait sur le jeu de données PlantVillage (voir annexes) et les images correspondent au sommaire des différentes couches. Ces sommaires sont produits grâce à torchsummary.

#### a) Modèle simple

Premièrement, j'ai construit mon tout premier modèle convolutionnel de façon très basique ; plusieurs blocs avec à chaque fois une couche de convolution, de la non-linéarité puis du Max Pooling. A la fin, on aplatis avec Flatten afin de pouvoir utiliser Linear et finalement avoir les prédictions.

*Figure 18 : Modèle « SimpleCNN »*

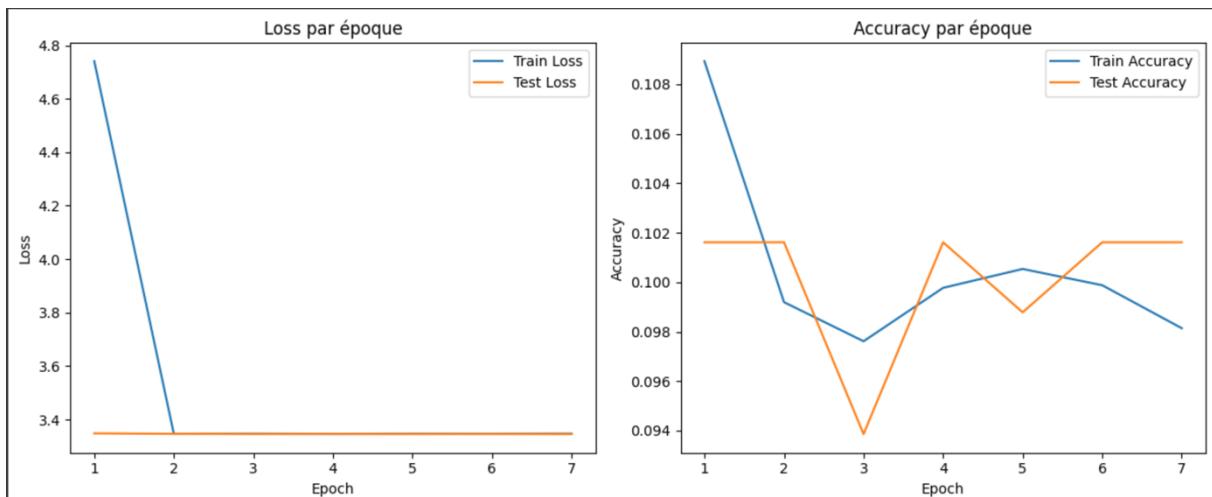
Layer (type)	Output Shape	Param #
Conv2d-1	[−1, 32, 256, 256]	896
ReLU-2	[−1, 32, 256, 256]	0
MaxPool2d-3	[−1, 32, 128, 128]	0
Conv2d-4	[−1, 64, 128, 128]	18,496
ReLU-5	[−1, 64, 128, 128]	0
MaxPool2d-6	[−1, 64, 64, 64]	0
Conv2d-7	[−1, 128, 64, 64]	73,856
ReLU-8	[−1, 128, 64, 64]	0
MaxPool2d-9	[−1, 128, 32, 32]	0
Conv2d-10	[−1, 256, 32, 32]	295,168
ReLU-11	[−1, 256, 32, 32]	0
MaxPool2d-12	[−1, 256, 16, 16]	0
Conv2d-13	[−1, 512, 16, 16]	1,180,160
ReLU-14	[−1, 512, 16, 16]	0
MaxPool2d-15	[−1, 512, 8, 8]	0
Flatten-16	[−1, 32768]	0
Linear-17	[−1, 512]	16,777,728
ReLU-18	[−1, 512]	0
Linear-19	[−1, 38]	19,494

➤ En l'entraînant, on se rend compte dès les premiers tours que les prédictions sont aléatoires, ne dépassant pas les 10% de précision même après 20 epochs (tour de boucle).

➤ Nombre de paramètres : 18 365 798

Source : Production personnelle

*Figure 19 : Évolution de la perte et de la précision au fil des epoch lors de l'entraînement*



Source : Production personnelle

Le modèle n'apprend rien et fais seulement des propositions aléatoires.

## b) Modèle très profond

Après avoir essayé de rajouter de la complexité dans le modèle précédent sans succès, j'ai également rajouté de la BatchNorm et du Dropout :

Figure 20 : Modèle « DeepCNN »

Conv2d-1	$[-1, 32, 256, 256]$	896
BatchNorm2d-2	$[-1, 32, 256, 256]$	64
ReLU-3	$[-1, 32, 256, 256]$	0
Conv2d-4	$[-1, 32, 256, 256]$	9,248
BatchNorm2d-5	$[-1, 32, 256, 256]$	64
ReLU-6	$[-1, 32, 256, 256]$	0
MaxPool2d-7	$[-1, 32, 128, 128]$	0
Conv2d-8	$[-1, 64, 128, 128]$	18,496
BatchNorm2d-9	$[-1, 64, 128, 128]$	128
ReLU-10	$[-1, 64, 128, 128]$	0
Conv2d-11	$[-1, 64, 128, 128]$	36,928
BatchNorm2d-12	$[-1, 64, 128, 128]$	128
ReLU-13	$[-1, 64, 128, 128]$	0
MaxPool2d-14	$[-1, 64, 64, 64]$	0
Conv2d-15	$[-1, 128, 64, 64]$	73,856
BatchNorm2d-16	$[-1, 128, 64, 64]$	256
ReLU-17	$[-1, 128, 64, 64]$	0
Conv2d-18	$[-1, 128, 64, 64]$	147,584
BatchNorm2d-19	$[-1, 128, 64, 64]$	256
ReLU-20	$[-1, 128, 64, 64]$	0
MaxPool2d-21	$[-1, 128, 32, 32]$	0
Conv2d-22	$[-1, 256, 32, 32]$	295,168
BatchNorm2d-23	$[-1, 256, 32, 32]$	512
ReLU-24	$[-1, 256, 32, 32]$	0
Conv2d-25	$[-1, 256, 32, 32]$	590,080
BatchNorm2d-26	$[-1, 256, 32, 32]$	512
ReLU-27	$[-1, 256, 32, 32]$	0
MaxPool2d-28	$[-1, 256, 16, 16]$	0
Conv2d-29	$[-1, 512, 16, 16]$	1,180,168
BatchNorm2d-30	$[-1, 512, 16, 16]$	1,024
ReLU-31	$[-1, 512, 16, 16]$	0
Conv2d-32	$[-1, 512, 16, 16]$	2,359,888
BatchNorm2d-33	$[-1, 512, 16, 16]$	1,024
ReLU-34	$[-1, 512, 16, 16]$	0
MaxPool2d-35	$[-1, 512, 8, 8]$	2,359,888
Conv2d-36	$[-1, 512, 8, 8]$	1,024
BatchNorm2d-37	$[-1, 512, 8, 8]$	0
ReLU-38	$[-1, 512, 8, 8]$	0
Conv2d-39	$[-1, 512, 8, 8]$	2,359,888
BatchNorm2d-40	$[-1, 512, 8, 8]$	1,024
ReLU-41	$[-1, 512, 8, 8]$	0
MaxPool2d-42	$[-1, 512, 4, 4]$	0
Conv2d-43	$[-1, 512, 4, 4]$	2,359,888
BatchNorm2d-44	$[-1, 512, 4, 4]$	1,024
ReLU-45	$[-1, 512, 4, 4]$	0
Conv2d-46	$[-1, 512, 4, 4]$	2,359,888
BatchNorm2d-47	$[-1, 512, 4, 4]$	1,024
ReLU-48	$[-1, 512, 4, 4]$	0
MaxPool2d-49	$[-1, 512, 2, 2]$	0
Conv2d-50	$[-1, 512, 2, 2]$	2,359,888
BatchNorm2d-51	$[-1, 512, 2, 2]$	1,024
ReLU-52	$[-1, 512, 2, 2]$	0
Conv2d-53	$[-1, 512, 2, 2]$	2,359,888
BatchNorm2d-54	$[-1, 512, 2, 2]$	1,024
ReLU-55	$[-1, 512, 2, 2]$	0
MaxPool2d-56	$[-1, 512, 1, 1]$	0
Flatten-57	$[-1, 512]$	0
Linear-58	$[-1, 1024]$	525,312
ReLU-59	$[-1, 1024]$	0
Dropout-60	$[-1, 1024]$	0
Linear-61	$[-1, 512]$	524,888
ReLU-62	$[-1, 512]$	0
Dropout-63	$[-1, 512]$	0
Linear-64	$[-1, 38]$	19,494

Source : Production personnelle

Ainsi j'ai dû implémenter mes premiers réseaux avec des connexions résiduelles pour résoudre ce problème.

➤ La précision augmente jusqu'à environ 25%, mais avec autant de couches le gradient ne peut pas bien circuler : c'est le problème du vanishing gradient.

➤ On comprend donc que la régularisation que ces nouveaux algorithmes apportent est efficace, mais un problème plus grand persiste

➤ Nombre de paramètres : 19 950 790

## c) Petit modèle résiduel : utilité de la data augmentation

Grâce aux blocs résiduels, mes nouveaux modèles performaient maintenant très bien, les prédictions étaient aux alentours de 98% de précision. Cependant, en visualisant les courbes d'entraînements ci-dessous j'ai redouté que le modèle commençait à faire de l'overfitting et à ne plus performer en test.

J'ai ainsi implémenté de la data augmentation afin de faire varier les images d'entraînement et réduire ce phénomène, pour que mon modèle soit plus stable. On peut observer une nette différence de la courbe de test.

Figure 21 : Modèle « SmallResNet »

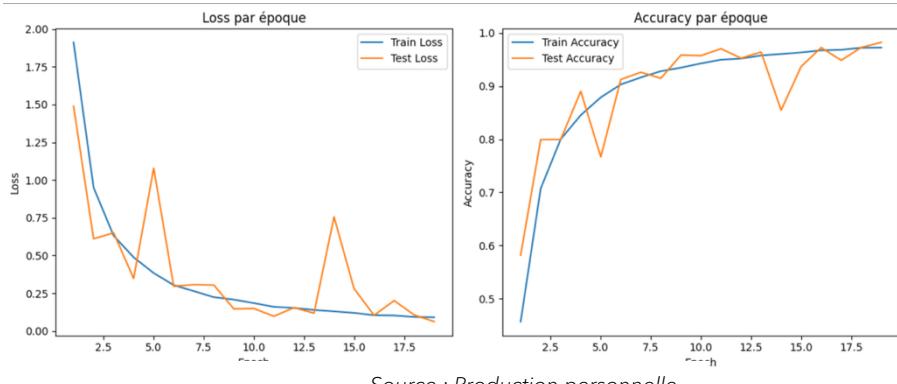
Layer (type)	Output Shape	Param #
Conv2d-1	[1, 32, 256, 256]	896
BatchNorm2d-2	[1, 32, 256, 256]	64
ReLU-3	[1, 32, 256, 256]	0
MaxPool2d-4	[1, 32, 128, 128]	0
Conv2d-5	[1, 64, 128, 128]	2,112
BatchNorm2d-6	[1, 64, 128, 128]	128
Conv2d-7	[1, 64, 128, 128]	18,496
BatchNorm2d-8	[1, 64, 128, 128]	128
ReLU-9	[1, 64, 128, 128]	0
Conv2d-10	[1, 64, 128, 128]	36,928
BatchNorm2d-11	[1, 64, 128, 128]	128
ReLU-12	[1, 64, 128, 128]	0
ResidualBlock_3-13	[1, 64, 128, 128]	0
MaxPool2d-14	[1, 64, 64, 64]	0
Conv2d-15	[1, 128, 64, 64]	8,320
BatchNorm2d-16	[1, 128, 64, 64]	256
Conv2d-17	[1, 128, 64, 64]	73,856
BatchNorm2d-18	[1, 128, 64, 64]	256
ReLU-19	[1, 128, 64, 64]	0
Conv2d-20	[1, 128, 64, 64]	147,584
BatchNorm2d-21	[1, 128, 64, 64]	256
ReLU-22	[1, 128, 64, 64]	0
ResidualBlock_3-23	[1, 128, 64, 64]	0
MaxPool2d-24	[1, 128, 32, 32]	0
Conv2d-25	[1, 256, 32, 32]	33,024
BatchNorm2d-26	[1, 256, 32, 32]	512
Conv2d-27	[1, 256, 32, 32]	295,168
BatchNorm2d-28	[1, 256, 32, 32]	512
ReLU-29	[1, 256, 32, 32]	0
Conv2d-30	[1, 256, 32, 32]	590,080
BatchNorm2d-31	[1, 256, 32, 32]	512
ReLU-32	[1, 256, 32, 32]	0
ResidualBlock_3-33	[1, 256, 32, 32]	0
MaxPool2d-34	[1, 256, 16, 16]	0
Conv2d-35	[1, 512, 16, 16]	131,594
BatchNorm2d-36	[1, 512, 16, 16]	1,024
Conv2d-37	[1, 512, 16, 16]	1,180,160
BatchNorm2d-38	[1, 512, 16, 16]	1,024
ReLU-39	[1, 512, 16, 16]	0
Conv2d-40	[1, 512, 16, 16]	2,359,808
BatchNorm2d-41	[1, 512, 16, 16]	1,024
ReLU-42	[1, 512, 16, 16]	0
ResidualBlock_3-43	[1, 512, 16, 16]	0
AdaptiveAvgPool2d-44	[1, 512, 1, 1]	0
Flatten-45	[1, 512]	0
Linear-46	[1, 256]	131,328
ReLU-47	[1, 256]	0
Dropout-48	[1, 256]	0
Linear-49	[1, 38]	9,766

Source : Production personnelle

➤ Jusqu'à 98-98.5% de précision, une grande amélioration grâce aux connexions résiduelles !

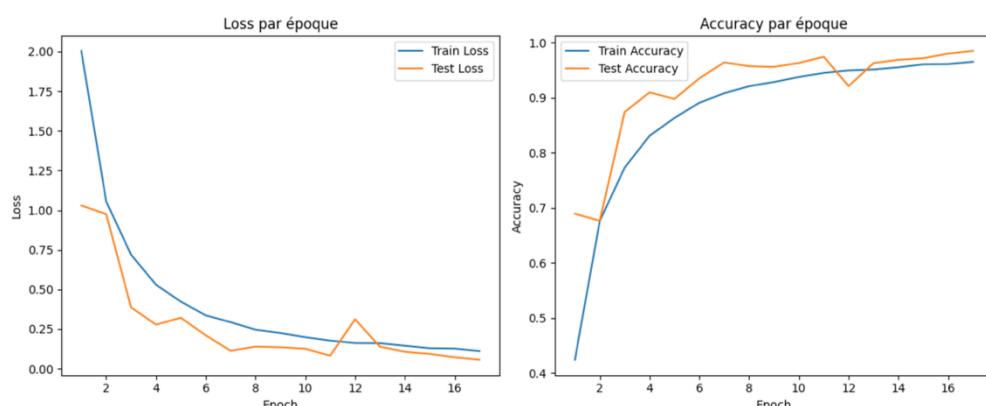
➤ Nombre de paramètres : 5,024,934

Figure 22 : Résultats de l'entraînement sans la data augmentation (98.20% accuracy)



➤ Peu stable

Figure 23 : Résultats de l'entraînement avec la data augmentation (98.46% accuracy)



➤ Plus stable

#### d) Modèle résiduel : stride = 2 comparé à MaxPool2d

Par la suite, parmi d'autres nombreuses expérimentations, j'ai remarqué que la différence entre un stride de 2 (un pas de 2 lors du glissement du filtre sur l'image) et du Max Pooling est considérable. Dans les faits, ils ont un impact similaire car ils permettent tous les deux de faire du downsampling c'est-à-dire de réduire la taille des images. Or, lorsqu'on compare au niveau des résultats c'est très différent :

Figure 24 : Modèle « ResNet » avec MaxPool

Layer (type)	Output Shape	Param #
Conv2d-1	[-, 32, 256, 256]	896
BatchNorm2d-2	[-, 32, 256, 256]	64
ReLU-3	[-, 32, 256, 256]	0
Conv2d-4	[-, 64, 256, 256]	2,112
BatchNorm2d-5	[-, 64, 256, 256]	128
Conv2d-6	[-, 64, 256, 256]	18,496
BatchNorm2d-7	[-, 64, 256, 256]	128
ReLU-8	[-, 64, 256, 256]	0
Conv2d-9	[-, 64, 256, 256]	36,928
BatchNorm2d-10	[-, 64, 256, 256]	128
ReLU-11	[-, 64, 256, 256]	0
ResidualBlock_2-12	[-, 64, 256, 256]	0
MaxPool2d-13	[-, 64, 128, 128]	0
Conv2d-14	[-, 128, 128, 128]	8,320
BatchNorm2d-15	[-, 128, 128, 128]	256
Conv2d-16	[-, 128, 128, 128]	73,856
BatchNorm2d-17	[-, 128, 128, 128]	256
ReLU-18	[-, 128, 128, 128]	0
Conv2d-19	[-, 128, 128, 128]	147,584
BatchNorm2d-20	[-, 128, 128, 128]	256
ReLU-21	[-, 128, 128, 128]	0
ResidualBlock_2-22	[-, 128, 128, 128]	0
MaxPool2d-23	[-, 128, 64, 64]	0
Conv2d-24	[-, 256, 64, 64]	33,024
BatchNorm2d-25	[-, 256, 64, 64]	512
Conv2d-26	[-, 256, 64, 64]	295,168
BatchNorm2d-27	[-, 256, 64, 64]	512
ReLU-28	[-, 256, 64, 64]	0
Conv2d-29	[-, 256, 64, 64]	590,080
BatchNorm2d-30	[-, 256, 64, 64]	512
ReLU-31	[-, 256, 64, 64]	0
ResidualBlock_2-32	[-, 256, 64, 64]	0
MaxPool2d-33	[-, 256, 32, 32]	0
Conv2d-34	[-, 512, 32, 32]	131,584
BatchNorm2d-35	[-, 512, 32, 32]	1,024
Conv2d-36	[-, 512, 32, 32]	1,180,160
BatchNorm2d-37	[-, 512, 32, 32]	1,024
ReLU-38	[-, 512, 32, 32]	0
Conv2d-39	[-, 512, 32, 32]	2,359,888
BatchNorm2d-40	[-, 512, 32, 32]	1,024
ReLU-41	[-, 512, 32, 32]	0
ResidualBlock_2-42	[-, 512, 32, 32]	0
MaxPool2d-43	[-, 512, 16, 16]	0
AdaptiveAvgPool2d-44	[-, 512, 1, 1]	0
Flatten-45	[-, 512]	0
Linear-46	[-, 512]	262,656
ReLU-47	[-, 512]	0
Dropout-48	[-, 512]	0
Linear-49	[-, 38]	19,494

➤ Avec MaxPool : Le temps d'exécution est de 20 mins / epoch soit 4 fois plus long comparé au stride = 2

➤ Les résultats restent similaires (= 98%)

➤ Nombre de paramètres : 5,165,990

Source : Production personnelle

Figure 25 : Modèle « ResNet » avec stride = 2

Layer (type)	Output Shape	Param #
Conv2d-1	[-, 32, 256, 256]	896
BatchNorm2d-2	[-, 32, 256, 256]	64
ReLU-3	[-, 32, 256, 256]	0
Conv2d-4	[-, 64, 128, 128]	2,112
BatchNorm2d-5	[-, 64, 128, 128]	128
Conv2d-6	[-, 64, 128, 128]	18,496
BatchNorm2d-7	[-, 64, 128, 128]	128
ReLU-8	[-, 64, 128, 128]	0
Conv2d-9	[-, 64, 128, 128]	36,928
BatchNorm2d-10	[-, 64, 128, 128]	128
ReLU-11	[-, 64, 128, 128]	0
ResidualBlock-12	[-, 64, 128, 128]	0
Conv2d-13	[-, 128, 64, 64]	8,320
BatchNorm2d-14	[-, 128, 64, 64]	256
Conv2d-15	[-, 128, 64, 64]	73,856
BatchNorm2d-16	[-, 128, 64, 64]	256
ReLU-17	[-, 128, 64, 64]	0
Conv2d-18	[-, 128, 64, 64]	147,584
BatchNorm2d-19	[-, 128, 64, 64]	256
ReLU-20	[-, 128, 64, 64]	0
ResidualBlock-21	[-, 128, 64, 64]	0
Conv2d-22	[-, 256, 32, 32]	33,024
BatchNorm2d-23	[-, 256, 32, 32]	512
Conv2d-24	[-, 256, 32, 32]	295,168
BatchNorm2d-25	[-, 256, 32, 32]	512
ReLU-26	[-, 256, 32, 32]	0
Conv2d-27	[-, 256, 32, 32]	590,080
BatchNorm2d-28	[-, 256, 32, 32]	512
ReLU-29	[-, 256, 32, 32]	0
ResidualBlock-30	[-, 256, 32, 32]	0
Conv2d-31	[-, 512, 16, 16]	131,584
BatchNorm2d-32	[-, 512, 16, 16]	1,024
Conv2d-33	[-, 512, 16, 16]	1,180,160
BatchNorm2d-34	[-, 512, 16, 16]	1,024
ReLU-35	[-, 512, 16, 16]	0
Conv2d-36	[-, 512, 16, 16]	2,359,888
BatchNorm2d-37	[-, 512, 16, 16]	1,024
ReLU-38	[-, 512, 16, 16]	0
ResidualBlock-39	[-, 512, 16, 16]	0
AdaptiveAvgPool2d-40	[-, 512, 1, 1]	0
Flatten-41	[-, 512]	0
Linear-42	[-, 512]	262,656
ReLU-43	[-, 512]	0
Dropout-44	[-, 512]	0
Linear-45	[-, 38]	19,494

➤ Avec stride = 2 : Temps d'exécution de 5 mins / epoch soit 4 fois plus court qu'avec le MaxPool

➤ Résultats similaires (= 98%)

➤ Nombre de paramètres : 5,165,990

Source : Production personnelle

J'ai ainsi pu m'intéresser à l'efficacité mémoire, et j'ai remarqué que la mémoire utilisée par le MaxPool est plus de 3 fois plus grande que le stride de 2 :

	<b>MaxPool</b>	<b>Stride = 2</b>
<b>Input Size (MB)</b>	0.75	0.75
<b>Forward/backward (MB)</b>	603.02	183.02
<b>Params (MB)</b>	19.71	19.71
<b>Total (MB)</b>	623.48	203.48

### e) Optimisation du modèle résiduel

Finalement, mon modèle le plus développé est un peu moins profond que les autres (j'ai retiré un bloc résiduel) mais possède beaucoup d'algorithmes de régularisation (LR Scheduler, weight decay, gradient clipping), ce qui m'a permis d'atteindre un peu plus de 99% de précision tout en réduisant le nombre de paramètres considérablement.

Figure 26 : Modèle « ResNet\_minimal »

Layer (type)	Output Shape	Param #	
Conv2d-1	[-, 32, 256, 256]	896	
BatchNorm2d-2	[-, 32, 256, 256]	64	
ReLU-3	[-, 32, 256, 256]	0	
Conv2d-4	[-, 64, 128, 128]	2,112	
BatchNorm2d-5	[-, 64, 128, 128]	128	
Conv2d-6	[-, 64, 128, 128]	18,496	
BatchNorm2d-7	[-, 64, 128, 128]	128	
ReLU-8	[-, 64, 128, 128]	0	
Conv2d-9	[-, 64, 128, 128]	36,928	
BatchNorm2d-10	[-, 64, 128, 128]	128	
ReLU-11	[-, 64, 128, 128]	0	
ResidualBlock-12	[-, 64, 128, 128]	0	
Conv2d-13	[-, 128, 64, 64]	8,320	➤ La précision augmente jusqu'à 99.02%
BatchNorm2d-14	[-, 128, 64, 64]	256	
Conv2d-15	[-, 128, 64, 64]	73,856	
BatchNorm2d-16	[-, 128, 64, 64]	256	
ReLU-17	[-, 128, 64, 64]	0	
Conv2d-18	[-, 128, 64, 64]	147,584	
BatchNorm2d-19	[-, 128, 64, 64]	256	
ReLU-20	[-, 128, 64, 64]	0	
ResidualBlock-21	[-, 128, 64, 64]	0	
Conv2d-22	[-, 256, 32, 32]	33,024	
BatchNorm2d-23	[-, 256, 32, 32]	512	
Conv2d-24	[-, 256, 32, 32]	295,168	
BatchNorm2d-25	[-, 256, 32, 32]	512	
ReLU-26	[-, 256, 32, 32]	0	
Conv2d-27	[-, 256, 32, 32]	590,080	
BatchNorm2d-28	[-, 256, 32, 32]	512	
ReLU-29	[-, 256, 32, 32]	0	
ResidualBlock-30	[-, 256, 32, 32]	0	
AdaptiveAvgPool2d-31	[-, 256, 1, 1]	0	
Flatten-32	[-, 256]	0	
Dropout-33	[-, 256]	0	
Linear-34	[-, 38]	9,766	

Source : Production personnelle

## IV. Conclusion

Ce stage m'a permis d'explorer les concepts fondamentaux des réseaux de neurones, avec une attention particulière portée à la convolution. Les compétences acquises notamment avec PyTorch m'ont permis de pouvoir expérimenter librement. C'est dans cette approche expérimentale que j'ai pu confronter la théorie à la pratique à travers la construction et l'optimisation d'un modèle. De nombreux axes d'améliorations sont apparus au cours du travail et il en existe toujours : réduction du nombre de paramètres, approfondissement des techniques de régularisation, architectures de modèles plus avancées... Par manque de temps, certains des nombreux algorithmes alternatifs n'ont pas pu être testés. Cependant, la modularité mise en place lors de mes développements permettrait de les intégrer facilement dans la suite du projet.

Cela met ainsi en lumière l'importance de la rigueur et de l'expérimentation dans la conception de modèles, et également l'amplitude des outils et ressources disponibles de nos jours pour se former sur l'IA et plus précisément en Deep Learning.

## V. Annexes

Jeu de données PlantVillage :  
<https://www.kaggle.com/datasets/abdallahhalidev/plantvillage-dataset>

### Ressources d'apprentissage en auto-formation :

- Deep Learning : <https://www.learnpytorch.io/>
- Machine Learning : Apprendre le Machine Learning en une semaine - Guillaume SAINT-CIRGUE
- ChatGPT
- Documentation PyTorch
- YouTube :
  - Dérivées : [https://www.youtube.com/watch?v=RLEE-iSBimc&https://www.youtube.com/watch?v=\\_QOIABG\\_OEo&t=225s](https://www.youtube.com/watch?v=RLEE-iSBimc&https://www.youtube.com/watch?v=_QOIABG_OEo&t=225s)
  - Tenseurs : [https://www.youtube.com/watch?v=zAGWR2\\_mUSMhttps://www.youtube.com/watch?v=f5liqUk0ZTw](https://www.youtube.com/watch?v=zAGWR2_mUSMhttps://www.youtube.com/watch?v=f5liqUk0ZTw)
  - Gradient : [https://www.youtube.com/watch?v=rcl\\_YRyoLIY&](https://www.youtube.com/watch?v=rcl_YRyoLIY&)
  - CNN : <https://www.youtube.com/watch?v=oGpzWAIP5p0&>
  - <https://www.youtube.com/watch?v=iUGTPcyYCPM&>