programmation de programmation 2

L3 informatique
HAI606I 2023-2024

Amusons-nous avec des images !

Réalisé par :

Youssef Grari
Arthur Conde Salazar
Naif Asswiel
Ilhame Ouhaddou

encadré par :

M. Pascal Poncelet

UNIVERSITÉ
DE MONTPELLIER

Table des matières

Introduction

Le domaine de l'apprentissage automatique offre d'innombrables possibilités pour repousser les frontières de l'intelligence artificielle (IA). Dans le cadre de ce projet, mené sous la supervision de **Monsieur Pascal Poncelet**, nous avons entrepris le développement d'une intelligence artificielle capable de reconnaître et de manipuler des images d'animaux tels que les tigres, les éléphants et les renards. Notre objectif principal consiste à mettre en œuvre un réseau de neurones convolutif (CNN) afin d'accomplir ces tâches avec précision et efficacité.

Le cœur de notre travail réside non seulement dans l'entraînement du modèle à reconnaître ces animaux dans des images mais aussi à obtenir des résultats fiables. Afin de garantir cette fiabilité, nous travaillerons d'abord sur nos modèles en essayant de les complexifier, puis nous nous pencherons sur la quantité et la qualité de notre jeu de données. Nous explorerons durant ce processus différentes pratiques utilisées par les leaders de ce domaine, telles que openAI.

Ce projet a été développé en Python en utilisant Google Colab [1], une plateforme de développement en ligne basée sur Jupyter Notebook. Google Colab offre un environnement de cloud computing permettant l'exécution de code Python. Il permet également un accès aux ressources matérielles telles que les GPU (Graphics Processing Units) et les TPU (Tensor Processing Units) de Google. Étant donné que les calculs effectués sont complexes et prennent beaucoup de temps, ces accélérateurs matériels sont indispensables pour garantir des temps d'exécution raisonnables lors de l'entraînement de modèles d'apprentissage automatique, et ce, sans nécessiter de matériel coûteux.

Le reste du mémoire est organisé de la manière suivante : dans le Chapitre 1, nous proposons un rappel des concepts de base qui seront utiles à la compréhension des travaux réalisés. Ainsi, nous rappelons les principes de l'apprentissage supervisé et présentons les concepts associés aux réseaux de neurones et aux convolutions. Le Chapitre 2 présente les travaux menés pour rechercher les meilleurs modèles de classification pour nos données. Ensuite, dans le Chapitre 3, nous aborderons la génération de nouvelles images.

Chapitre 1

Bases de l'Apprentissage Automatique pour la Classification

Dans le cadre de nos travaux, notre attention s'est principalement portée sur l'apprentissage automatique et plus particulièrement sur la classification supervisée. Ainsi, nous débuterons en proposant un rappel des principes fondamentaux qui y sont associés. Étant donné que notre contexte concerne la classification d'images, dans un premier temps, nous rappelons les concepts de base des réseaux de neurones avant d'aborder spécifiquement les réseaux de convolution.

1.1 Classification

L'apprentissage automatique représente un domaine majeur de l'intelligence artificielle, visant à permettre à une machine d'apprendre et d'améliorer ses performances à partir de données grâce à des méthodes mathématiques. Dans nos travaux, nous nous concentrons principalement sur la classification supervisée, qui requiert un ensemble de données étiquetées. Par exemple, dans le cas de textes, les étiquettes peuvent indiquer si le texte est positif ou négatif, tandis que pour les images, les étiquettes spécifient la classe à laquelle appartient chaque image, telle que "femme" ou "homme".

Le processus de classification supervisée consiste à entraîner un modèle à partir d'un ensemble de données d'apprentissage afin de déterminer au mieux les caractéristiques distinctives de chaque classe. Par exemple, si la présence du mot "adore" est caractéristique des textes positifs, le modèle pourra prédire la classe positive lorsqu'il rencontre ce terme. L'évaluation de la qualité du modèle se fait généralement en utilisant un jeu de test distinct, où l'on connaît les véritables étiquettes, et en calculant des mesures telles que l'accuracy [5], qui compare le nombre de prédictions correctes avec le nombre total de prédictions, ou d'autres types de mesures moins intuitives [2].

Une fois que nous avons sélectionné un ensemble de données représentatif, nous passons à l'étape cruciale de la classification. Cela implique d'attribuer des étiquettes ou des catégories à nos données en fonction de leurs caractéristiques. Dans notre contexte, où nous traitons les images comme des vecteurs avec des étiquettes de classe, cette étape revêt une importance particulière.

Pour ce faire, nous utilisons un algorithme de classification capable de prédire la classe ou l'étiquette correspondante pour chaque donnée en sortie. Cet algorithme s'appuie sur un ensemble de données diversifié que nous lui fournissons.

Une fois l'algorithme choisi, nous procédons à une phase d'entraînement. Pendant cette phase, le modèle est exposé à un ensemble de données d'apprentissage préalablement étiquetées. Il ajuste ses paramètres internes pour minimiser les erreurs de prédiction sur cet

ensemble de données. Cette étape est essentielle car elle permet au modèle de s'adapter aux spécificités des données et d'améliorer sa capacité à généraliser et à faire des prédictions précises sur de nouvelles données.

Dans cette optique, nous divisons généralement notre ensemble de données en deux sous-ensembles distincts : le jeu d'apprentissage et le jeu de test. Le jeu d'apprentissage est utilisé pour entraîner le modèle, tandis que le jeu de test est utilisé pour évaluer ses performances sur des données qu'il n'a pas encore vues. Cela nous permet de vérifier la capacité du modèle à généraliser ses connaissances et à produire des résultats précis sur de nouvelles données.

Avant d'explorer en détails le processus de classification, jetons un coup d'œil à la Figure 1.1 ci-dessous. Elle nous offre une vue d'ensemble des étapes clés de ce processus, nous permettant ainsi de mieux appréhender sa complexité.

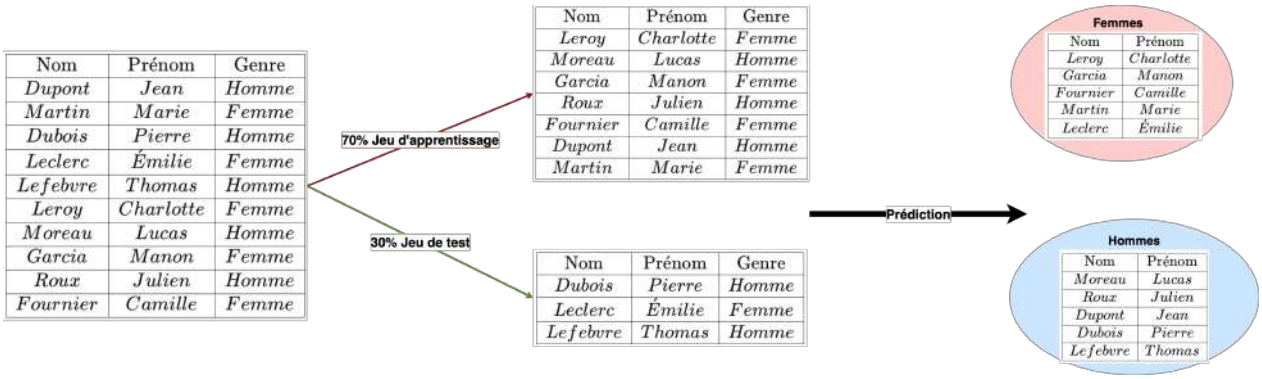

FIGURE 1.1 – Processus de Classification

Il existe de très nombreux classifieurs chacun ayant ses propres avantages et inconvénients [4]. Dans le cadre de nos travaux nous nous sommes principalement intéressés aux réseaux de neurones dont nous décrivons le fonctionnement dans la section suivante.

1.2 Réseaux de Neurones Convolutifs (CNN)

Les réseaux de neurones constituent une approche puissante largement utilisée dans le domaine de l'apprentissage automatique, notamment pour la classification de données. Cependant, lorsqu'il s'agit de traiter des images, le défi devient plus complexe en raison de la nature structurée des données. Contrairement aux données tabulaires, les images sont représentées sous forme de matrices bidimensionnelles plutôt que de vecteurs plats. C'est là que les réseaux de neurones convolutifs (CNN) entrent en jeu.

1.2.1 Réseaux de neurones

Un réseau de neurones est un modèle informatique sophistiqué composé de couches interconnectées de neurones. Ces couches agissent comme des unités de traitement, recevant des données en entrée et produisant des sorties en fonction des schémas appris au cours de l'entraînement. Typiquement, un réseau de neurones comprend trois types de couches : la couche d'entrée, les couches cachées et la couche de sortie. La couche d'entrée constitue le point d'entrée du réseau, chaque neurone représentant une caractéristique ou une variable d'entrée. Les couches cachées effectuent le traitement intermédiaire des données en recevant des signaux d'entrée pondérés, en effectuant des calculs et en appliquant des fonctions d'activation pour

introduire de la non-linéarité. Enfin, la couche de sortie produit les résultats finaux du réseau, chaque neurone correspondant à une classe ou une étiquette de sortie possible.

Le fonctionnement d'un réseau de neurones implique le forward propagation, où les valeurs prédictives sont multipliées par les poids des connexions, puis sommées et soumises à une fonction d'activation pour capturer les données complexes. Ensuite, la prédiction est comparée aux valeurs attendues à l'aide de fonctions de coût, et les erreurs sont propagées à travers le réseau via la Backward Propagation pour ajuster les poids. Cette itération permet au réseau de s'adapter progressivement aux données et d'améliorer ses performances.

La Figure 1.2 illustre les différentes couches du modèle ainsi que les poids initialisés de manière aléatoire. Nous observons que ce réseau comporte 3 neurones en entrée, correspondant aux variables prédictives. Il est constitué de deux couches cachées, tandis que la dernière couche ne possède qu'un seul neurone. Ce modèle est conçu pour effectuer une classification binaire.

NB : bien que la figure utilise une couche de sortie avec un seul neurone, cette architecture est adaptable pour des tâches de classification binaire, où une réponse binaire est nécessaire (0 pour non, 1 pour oui), mais peut être étendue pour des classifications multi-classes avec plusieurs neurones de sortie.

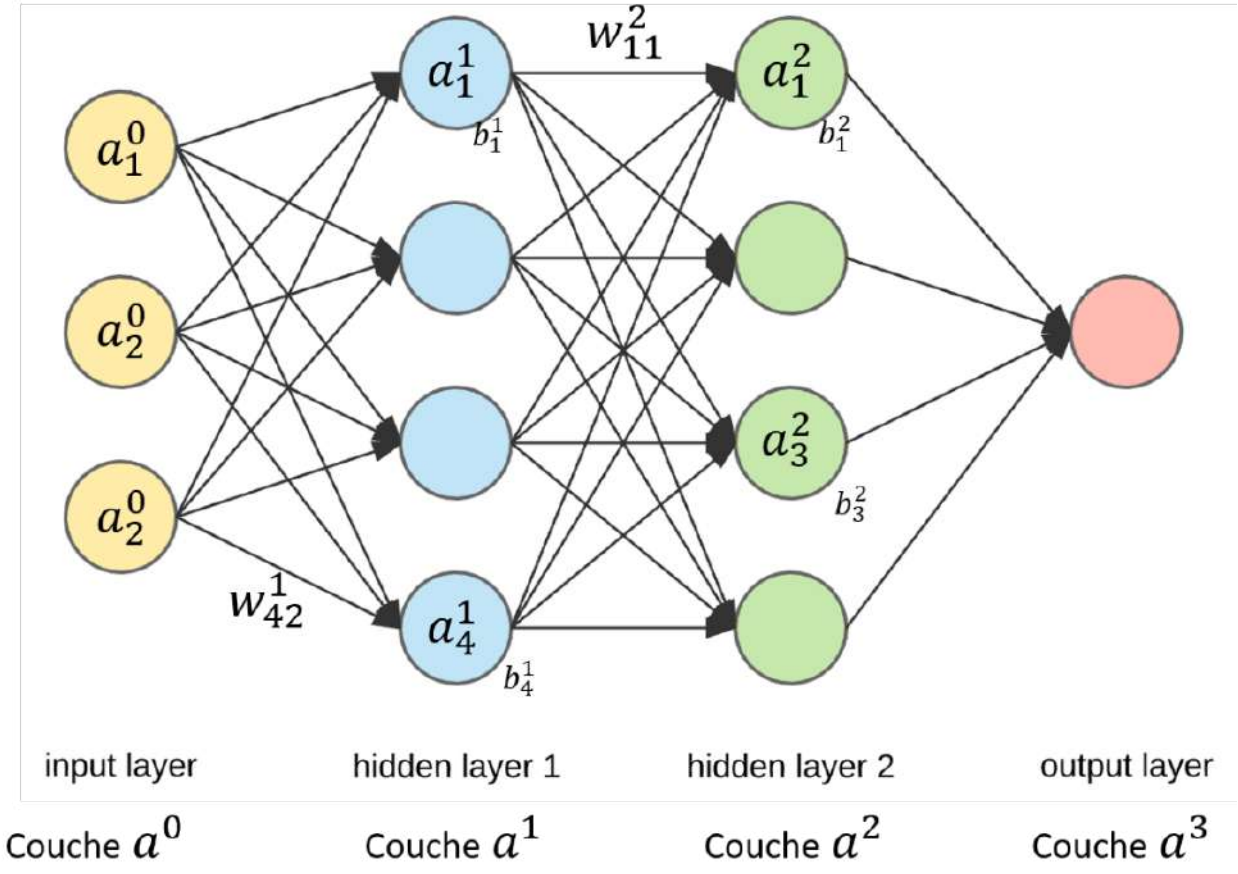

FIGURE 1.2 – Architecture d'un réseau de neurones

1.2.2 Convolution

Les pré-traitements des données d'entrée sont essentiels dans les réseaux de neurones, surtout pour les images. Contrairement à d'autres types de données, comme le texte, qui peuvent nécessiter des ajustements manuels, le traitement des images implique souvent des

pré-traitements complexes. Les opérations de convolution automatisent ces pré-traitements en extrayant automatiquement des caractéristiques visuelles, telles que les contours et les textures, sans intervention manuelle. Elle consiste à appliquer un Filtre, représenté par une matrice carrée de dimensions spécifiées, à la matrice d'origine représentant notre image. Cette opération génère une nouvelle matrice, souvent de taille réduite, qui contient des informations essentielles extraites de l'image d'entrée (extraction de features - *Feature Extraction*. Cette matrice résultante est ensuite soumise à une autre opération appelée *Pooling*, qui consiste à appliquer un autre Filtre pour réduire davantage la taille de l'image, tout en préservant ses caractéristiques importantes.Ces opérations de convolution et de pooling peuvent être répétées d'une manière récursive, permettant ainsi d'obtenir une image progressivement compressée tout en conservant les caractéristiques essentielles.

Une fois cette compression réalisée, l'image est aplatie (*Flatten*) pour être transformée en un vecteur, qui sera ensuite transmis à notre réseau de neurones pour le processus d'apprentissage. Ce processus, connu sous le nom de *Forward Propagation*, est suivi d'une étape cruciale appelée *Backward Propagation*. Pendant la Backward propagation, qui est le processus inverse de ce que nous venons de détailler, les paramètres du réseau, y compris les poids, sont ajustés en fonction des erreurs de prédiction, permettant ainsi au réseau de s'adapter et d'apprendre à mieux représenter les données.

La Figure 1.3 représente un réseau de neurones accompagné d'une couche de convolution en amont. On remarque que dans la première partie, les étapes de convolution sont appliquées pour extraire les caractéristiques des données. Ensuite, la couche de "flatten" est présente, permettant de convertir les caractéristiques extraites en un format adapté à l'entrée du réseau de neurones. Enfin, cette couche est suivie par la partie de classification réalisée par le réseau de neurones.

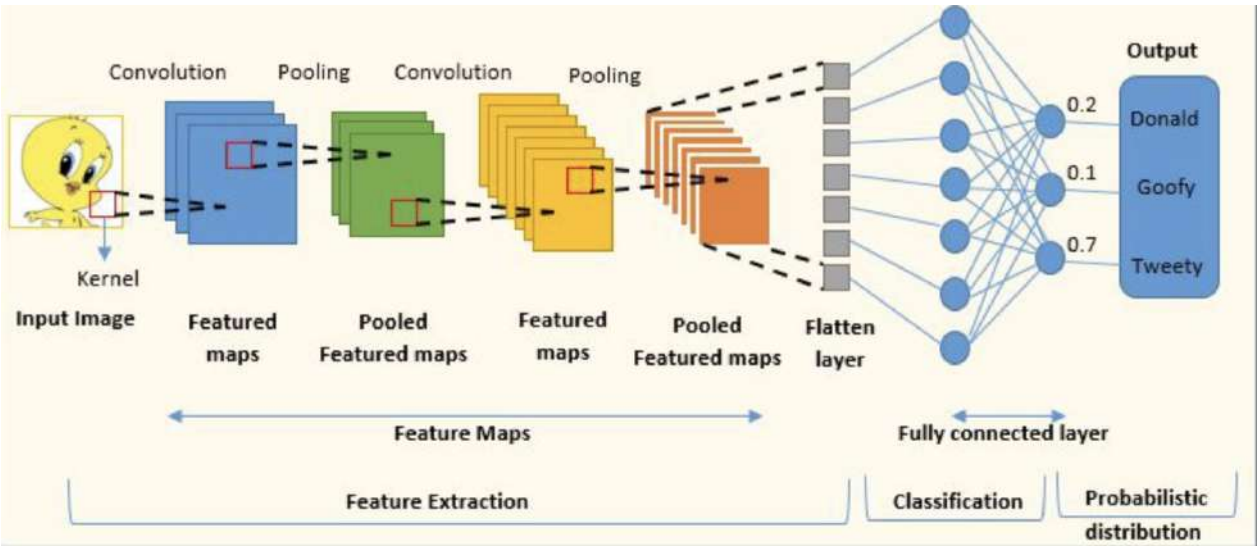

FIGURE 1.3 – Couches de convolution

Chapitre 2

Classification d'images

Dans ce chapitre consacré à la classification d'images, nous explorerons différentes approches et techniques pour améliorer la précision de nos modèles. Tout d'abord, nous aborderons la classification des données, en mettant l'accent sur l'importance de cette étape fondamentale dans le processus de l'apprentissage. Ensuite, nous nous pencherons sur des méthodes pour rendre nos modèles plus complexes afin d'améliorer nos résultats, notamment en ajoutant des couches supplémentaires et en modifiant les filtres des réseaux de neurones convolutifs.

Par la suite, nous étudierons l'impact de l'ajout de nouvelles données sur la performance de nos modèles, ainsi que l'utilisation du *Transfer Learning*, une technique puissante pour transférer les connaissances d'un modèle pré-entraîné vers un nouveau modèle pour des tâches similaires.

2.1 Description des données utilisées et pré-traitements

Pour élaborer notre modèle, nous avons constitué un ensemble de données comprenant trois catégories distinctes : les tigres, les éléphants et les renards, avec chacune 100 images. Nous avons également inclus 100 images négatives pour l'entraînement et l'évaluation du modèle dans le cadre de la classification.

Initialement, nos images présentent des variations de taille, ce qui peut influer sur la performance du modèle et compliquer le processus de traitement. De plus, les valeurs des pixels varient dans une large plage, allant de 0 à 255. Afin de pallier ces problèmes, nous avons procédé à une étape de *normalisation*.

Cela implique de diviser toutes les valeurs de pixels par 255, ramenant ainsi l'intervalle à une échelle de 0 à 1. De plus, nous avons redimensionné toutes nos images pour qu'elles possèdent des dimensions uniformes, facilitant ainsi leur traitement par notre algorithme.

La Figure 2.1 offre une représentation globale de notre jeu de données. Elle présente quelques exemples d'images positives et négatives pour chaque classe, utilisées pour l'entraînement du modèle.

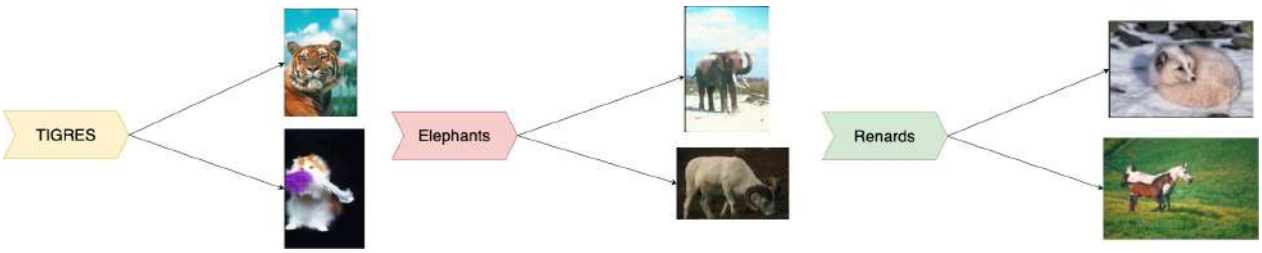

FIGURE 2.1 – Exemples d'images positives et négatives pour chaque classe

2.2 Classification des données

Dans un premier temps, notre objectif est de mettre en place un réseau de neurones de base, communément appelé *BaseLine*, qui servira de point de référence pour évaluer les améliorations et les extensions que nous envisageons par la suite. Ce réseau de base, caractérisé par sa simplicité, nous permettra de poser les fondations de notre approche et de comprendre les performances de base que nous cherchons à améliorer. En établissant ce point de départ, nous pourrons ensuite explorer de manière plus approfondie les différentes avenues pour optimiser et étendre notre modèle de classification d'images.

2.2.1 Modèle BaseLine

Nous commencerons par créer un modèle de base qui nous servira de point de départ pour développer des modèles plus sophistiqués susceptibles de produire de meilleurs résultats. Ce modèle de base comprendra une seule couche de convolution avec *16 filtres 3x3*, suivie de *deux couches de réseaux de neurones* comportant respectivement 100 neurones sur la couche d'entrée et 1 neurone sur la couche de sortie. Nous utiliserons les fonctions d'activation *'relu'* et *'sigmoid'* pour ces couches, et nous opterons pour l'optimiseur SGD afin de tracer nos courbes de perte *(loss)* [5] et d'exactitude *(accuracy)*, avec un taux d'apprentissage fixé à 0.01. Nous lancerons le modèle en utilisant une cross validation à 3 folds [5], tout en faisant varier le nombre d'epochs [5] et la batch size [5] lors de l'apprentissage. Le tableau ci-dessous illustre les résultats obtenus avec ce modèle de base :

Modèle Baseline							
Paramètres		**Éléphant**		**Tigre**		**Renard**	
Epochs	*batch*	*Moyenne*	*Écart type*	*Moyenne*	*Écart type*	*Moyenne*	*Écart type*
10	32	**83.4991**	**1.2242**	59.4451	8.8463	86.9817	6.8804
7	**16**	81.9614	5.7561	**67.4883**	**6.3133**	85.0143	12.8621
5	8	81.9840	5.3730	59.9276	10.2340	76.3078	17.6349
10	8	67.4204	11.6521	61.4729	4.0164	**94.9721**	**0.7307**

TABLE 2.1 – Performances du modèle Baseline

Analyse du tableau :

On peut remarquer que ce modèle est encore loin des performances souhaitées. En effet, il ne parvient pas à produire de bons résultats pour tous les animaux. La deuxième ligne, avec 7 époques et une taille de lot (*batch size*) de 16, présente la moyenne la plus élevée. Cependant, même dans ce cas, la meilleure précision (*accuracy*) n'est atteinte que pour la reconnaissance des tigres. Les éléphants pourraient atteindre une précision de 83.4991 et les renards 94.9721 avec ce même modèle.

La courbe de perte (C.f. Figure 2.2 (*loss*)) illustre l'évolution de l'erreur du modèle pendant l'apprentissage. Nous considérons ici la classe d'images pour laquelle notre modèle a donné les meilleurs résultats. Initialement, elle diminue régulièrement à mesure que le modèle apprend à partir des données d'entraînement, indiquant ainsi une amélioration de sa capacité à prédire les étiquettes correctes. Cependant, après un certain nombre d'époques, la courbe de perte commence à stagner ou même à augmenter, tandis que les performances du modèle sur les données de validation se dégradent. Ceci suggère un phénomène de surapprentissage (*overfitting*), où le modèle devient trop spécialisé dans les données d'entraînement spécifiques et perd sa capacité à généraliser à de nouvelles données.

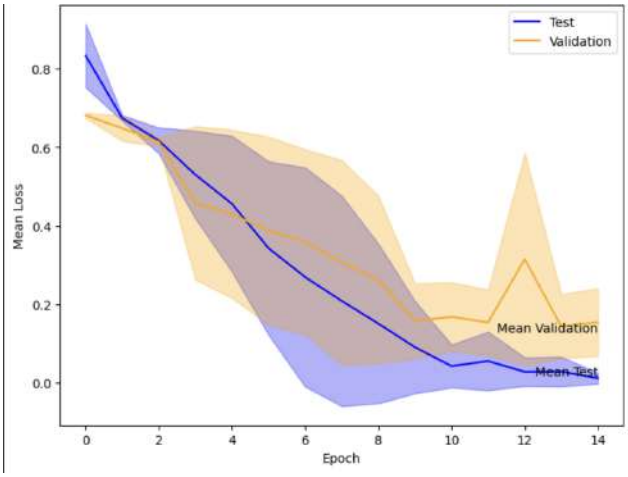

FIGURE 2.2 – Courbe de perte (Loss)

2.2.2 Vers des modèles plus complexes

En vue de corriger ces anomalies et d'améliorer les performances de notre modèle, nous envisageons de le rendre légèrement plus complexe en introduisant des modifications. Cela comprend notamment l'ajout de couches supplémentaires pour enrichir la capacité de représentation du modèle et l'adaptation des filtres pour capturer des caractéristiques plus précises et discriminantes dans les données. En optimisant la structure et les paramètres du modèle, nous visons à réduire le phénomène de surapprentissage et à améliorer sa capacité à généraliser à de nouvelles données.

Modification de couches

Dans le cadre de l'amélioration de notre modèle, nous avons introduit une technique appelée dropout. Le dropout est une méthode de régularisation utilisée dans les réseaux de neurones pour prévenir le surapprentissage. Concrètement, pendant l'entraînement du réseau, le dropout consiste à désactiver de manière aléatoire un certain pourcentage de neurones dans une couche donnée. Dans notre cas, nous avons appliqué un dropout de 40%. Cela signifie que pendant chaque itération d'entraînement, chaque neurone de la couche sur laquelle le dropout est appliqué a une probabilité de 40% d'être temporairement désactivé.

L'objectif du dropout est d'empêcher les neurones de devenir trop spécialisés ou dépendants les uns des autres pendant l'entraînement. En désactivant aléatoirement certains neurones à chaque itération, le dropout force le réseau à apprendre de manière plus robuste et à généraliser mieux sur de nouvelles données. Le tableau ci-dessous illustre les résultats

obtenus grâce à ces modifications :

Modèle Dropout							
Paramètres		**Éléphant**		**Tigre**		**Renard**	
Epochs	*batch*	*Moyenne*	*Écart type*	*Moyenne*	*Écart type*	*Moyenne*	*Écart type*
15	16	**83.997**	**1.887**	52.50	3.4999	89.499	5.50
15	8	66	20	53.499	3.499	**95.5**	**1.129**
8	64	72	7	68	1	62.58	3.5
8	32	69	15	**69**	**0.3**	55.499	3.499

TABLE 2.2 – Performances du modèle Dropout

Analyse du tableau :

Nous constatons que les performances obtenues avec le modèle amélioré ne divergent pas significativement de celles de notre précédent modèle. Cependant, une amélioration notable se manifeste au niveau de la réduction du phénomène de surapprentissage, attribuable à l'incorporation du Dropout. Cette observation est clairement illustrée dans la Figure 2.3 ci-dessous.

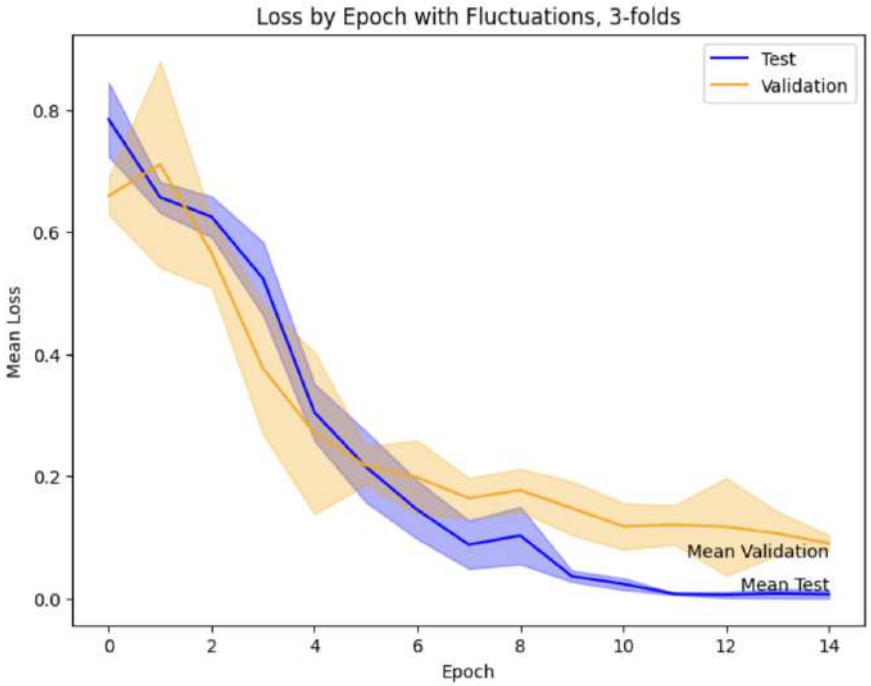

FIGURE 2.3 – Loss après le Dropout

2.2.3 Ajout de nouvelles données

Le dropout est souvent efficace pour atténuer le surapprentissage. Cependant, lorsque les données sont insuffisantes, cela peut encourager le modèle à mémoriser plutôt qu'à véritablement apprendre à généraliser. Pour contrer ce problème, une approche complémentaire consiste à augmenter la diversité des données en *générant des images* supplémentaires. Cette diversification élargit le spectre des motifs que le modèle peut apprendre, renforçant ainsi sa capacité à généraliser. Ainsi, en combinant le dropout avec la génération d'images, on adopte une stratégie prometteuse pour améliorer les performances du modèle tout en limitant le

surapprentissage.

Augmentation de données avec ImageDataGenerator

Pour accroître la quantité de données disponibles pour l'entraînement de notre modèle, chercher des centaines d'images sur internet serait une tâche titanesque. Heureusement, la classe *ImageDataGenerator* de la bibliothèque Keras offre une solution. Cette classe permet de générer de nouvelles images en appliquant diverses transformations aux images existantes, telles que des rotations, des zooms, des translations, des renversements horizontaux ou verticaux, et des ajustements de luminosité et de contraste. En appliquant ces transformations de manière aléatoire aux images d'origine, on crée un ensemble de données plus varié et plus robuste. Cela permet au modèle d'apprendre à reconnaître les mêmes motifs sous différentes formes, positions et orientations, ce qui améliore sa capacité à généraliser et à produire des prédictions précises sur de nouvelles données.

Cependant, il est essentiel de choisir judicieusement les transformations à appliquer et de les paramétrer de manière appropriée. Par exemple, il convient d'éviter les transformations qui pourraient induire des distorsions incohérentes avec la réalité, telles qu'une rotation verticale complète sur une image d'éléphant, qui pourrait conduire le modèle à apprendre des caractéristiques non pertinentes, il est assez rare de voir des éléphants marcher sur la tête en effet.

La Figure 2.4 présente quelques exemples d'images générées à partir de notre jeu de données à l'aide de ImageDataGenerator. Comme illustré, ces transformations incluent une rotation horizontale, un léger zoom et une modification du contraste. Malgré ces altérations, l'apparence générale de l'image originale est préservée, permettant ainsi d'enrichir l'apprentissage du modèle en lui exposant différentes positions de l'objet.

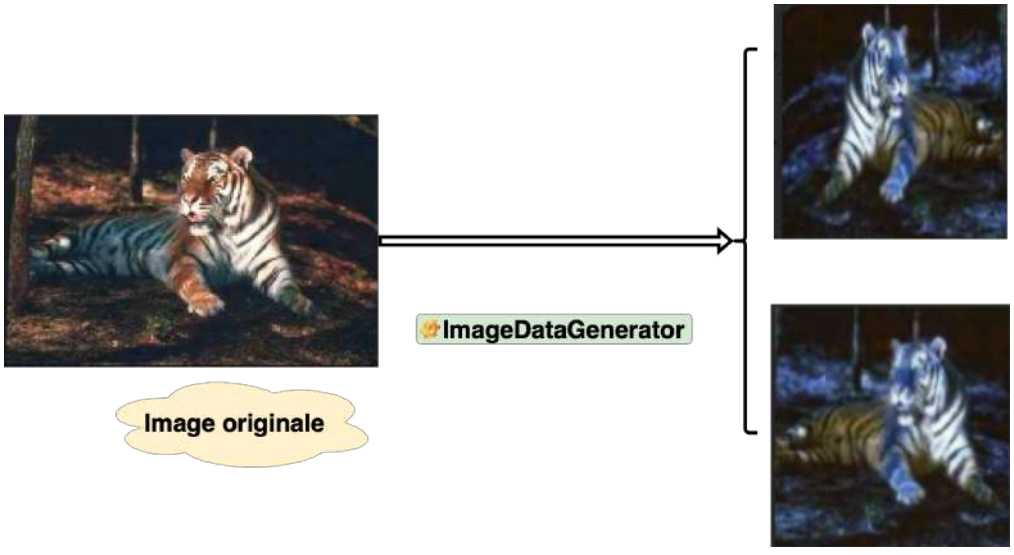

FIGURE 2.4 – Génération d'images avec ImageDataGenerator

Analyse des performances

Ces altérations n'ont pas eu un impact significatif sur les valeurs de précision, cependant, l'effet se fait sentir au niveau de nos courbes de perte. Comme le montre la Figure 2.5 ci-dessous, nous observons une réduction notable du phénomène de surapprentissage.

Ces variations introduites lors de la génération d'images ont permis au modèle de mieux généraliser en réduisant les écarts entre les performances sur les données d'entraînement et de validation.

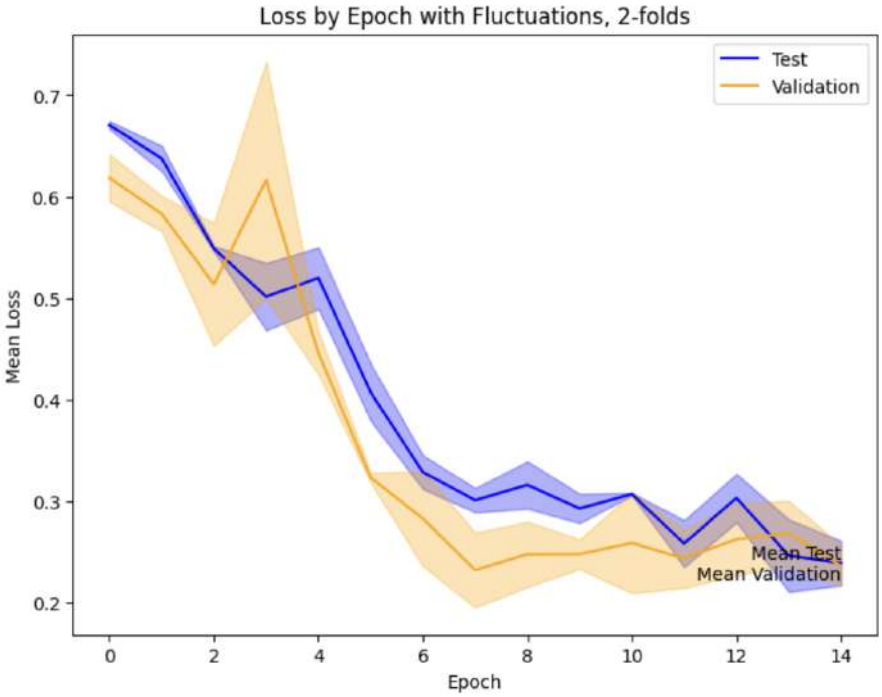

FIGURE 2.5 – Courbe de perte après utilisation de ImageDataGenerator

Retour sur l'accuracy et Feature Maps

Malgré les progrès réalisés dans l'atténuation du surapprentissage, notre modèle continue de présenter des déséquilibres dans les scores d'exactitude, comme le révèlent les Figures 2.2.1 et 2.2.2. Pour mieux comprendre cette disparité, plongeons dans le monde des *feature maps*.

Les feature maps sont des représentations des activations des filtres de convolution à travers différentes couches d'un réseau de neurones convolutif (CNN). Chaque carte de caractéristiques capture des informations spécifiques sur l'image en entrée. Lorsque l'image est propagée à travers le réseau, les filtres de convolution identifient des motifs et des caractéristiques visuelles telles que les bords, les textures ou les motifs plus complexes.

Lorsque nous examinons les feature maps d'une couche donnée, nous pouvons visualiser les zones de l'image qui ont activé les filtres de cette couche. Cela nous permet de comprendre quelles caractéristiques visuelles sont extraites à chaque étape du réseau. Par exemple, une feature map dans les premières couches peut montrer des activations correspondant à des bords ou des formes simples, tandis que dans les couches plus profondes, les feature maps peuvent représenter des motifs plus complexes et abstraits.

En analysant les feature maps de différentes couches, nous pouvons identifier les zones de l'image qui ont une influence prépondérante sur les prédictions du modèle. Cette analyse peut révéler des motifs ou des structures qui sont mal interprétés ou sous-représentés par le modèle, ce qui peut aider à diagnostiquer les problèmes de performance et à guider les ajustements du réseau pour améliorer la précision des prédictions.

Prenons deux images aléatoires de notre ensemble de données, l'une représentant un renard (Figure 2.8), et l'autre un tigre (Figure 2.6).

Commençons par examiner l'image du tigre et observons les résultats dans la Figure 2.7. Nous constatons que notre modèle éprouve des difficultés à identifier les pixels caractéristiques du tigre. L'arrière-plan présente une surabondance de textures et de couleurs, ce qui perturbe la caractérisation du tigre dans l'image.

D'autre part, en ce qui concerne l'image du renard, nous remarquons sur la Figure 2.9 que le modèle parvient à discriminer les contours très efficacement et à le différencier du fond sans problème apparent. Cette réussite peut être attribuée à la qualité et à la composition de l'image, où le fond présente peu de perturbations qui pourraient affecter l'apprentissage du modèle.

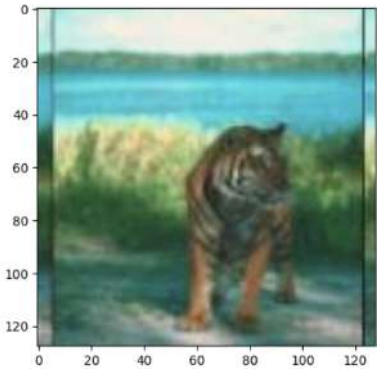

FIGURE 2.6 – Image prise au hasard d'un tigre

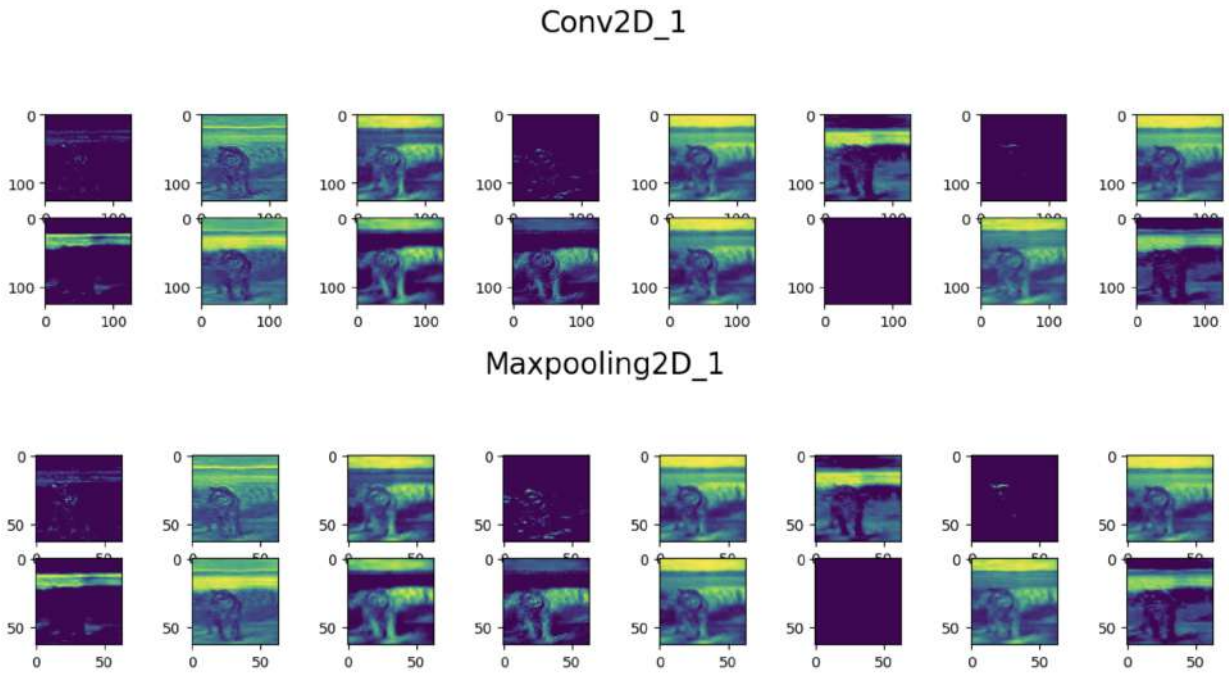

FIGURE 2.7 – Feature Maps Tigre

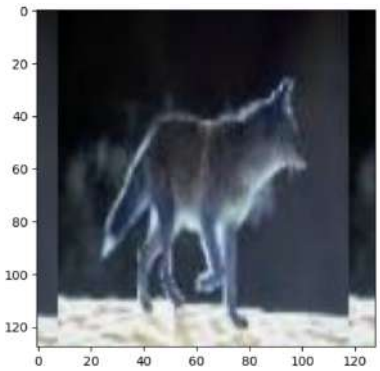

FIGURE 2.8 – Image prise au hasard d'un renard

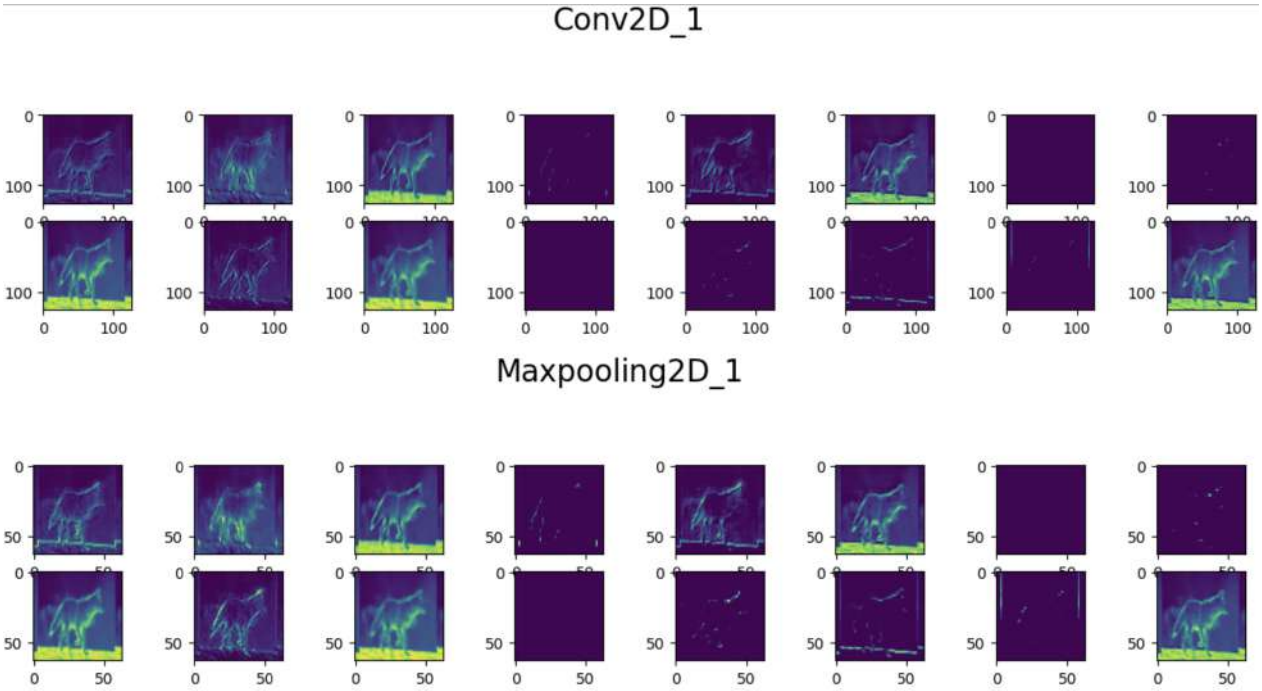

FIGURE 2.9 – Feature Maps Renard

Il est crucial de trouver une solution efficace pour résoudre ce déséquilibre. Notre objectif est que notre modèle puisse reconnaître avec précision les trois types d'animaux, malgré les variations de qualité des images. Si un être humain parvient à identifier facilement un animal dans une image, cette tâche ne devrait pas être trop difficile pour notre modèle. Pour atteindre cet objectif, nous allons recourir à une technique appelée *Transfer Learning*. Cette méthode nous permettra d'utiliser des modèles pré-entraînés et hautement performants pour enrichir notre propre modèle.

2.2.4 Transfer Learning

Dans cette section, nous introduisons la notion du transfer learning, une pratique qui nous permettra de compléter notre modèle afin d'améliorer ses performances. Le transfer learning consiste à utiliser des modèles pré-entraînés sur des tâches plus ou moins similaires pour résoudre de nouveaux problèmes, ce qui permet de capitaliser sur les connaissances déjà acquises et d'éviter de recommencer l'apprentissage à partir de zéro. Dans notre contexte,

nous explorerons l'utilisation du transfer learning avec le modèle *ResNet50* pour enrichir notre modèle existant et l'aider à mieux reconnaître les animaux dans nos images, malgré les variations de qualité et de couleur.

Utilisation du Transfer Learning avec ResNet50

Le transfer learning est une technique clé en apprentissage automatique qui consiste à utiliser des modèles pré-entraînés sur des tâches similaires pour résoudre de nouveaux problèmes. Plutôt que de construire un modèle à partir de zéro et d'apprendre des paramètres à partir de données brutes, le transfer learning permet de tirer parti des connaissances acquises par des modèles pré-entraînés sur des ensembles de données massives.

L'un des modèles pré-entraînés les plus populaires est ResNet50, une architecture de réseau neuronal profond proposée par Microsoft en 2015. ResNet50 est particulièrement réputé pour sa profondeur impressionnante, comprenant 48 couches de convolution, chacune suivie d'une couche de MaxPooling et d'AvaragePooling. Avec ses millions de paramètres à apprendre, ResNet50 est un modèle massif, mais ses performances exceptionnelles en font un choix idéal pour le transfer learning.

L'idée principale derrière le transfer learning avec ResNet50 est de "geler" les poids du modèle, ce qui signifie qu'ils ne seront pas mis à jour pendant l'entraînement (donc pas de Back Propagation). Ainsi, au lieu d'apprendre de nouveaux poids, nous utilisons les poids déjà mis à jour de ResNet50 comme point de départ. Ensuite, nous ajoutons des couches de classification personnalisées adaptées à notre tâche spécifique à la fin du modèle, ce qui nous permet d'adapter ResNet50 à notre problème particulier.

La Figure 2.10 ci-dessous illustre de manière caricaturale le processus de transfert de performances. Dans cette représentation, un robot dépourvu de connaissances s'enrichit en assimilant les neurones d'un autre robot, doté d'une intelligence plus développée.

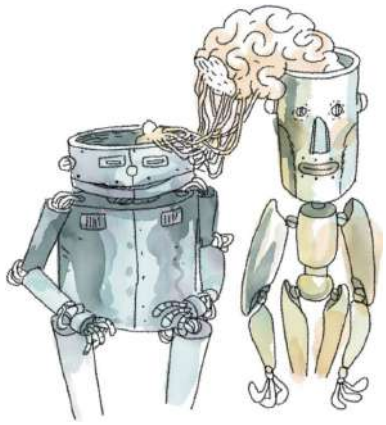

FIGURE 2.10 – Caricature décrivant le Transfer Learning

Discutons des résultats

La Table 2.2.2 illustre les performances de notre ancien modèle, où nous avons atteint des taux de précision de 95,5% pour les renards, 84% pour les éléphants et 69% pour les tigres dans les meilleures configurations. Cependant, avec l'introduction de notre nouveau modèle (roulement de tambours...), ces performances ont été surpassées, atteignant respectivement 97%, 95% et 94% pour les renards, les éléphants et les tigres.

De plus, nos courbes de perte démontrent une réduction significative du phénomène de surapprentissage par rapport à nos modèles précédents.

La Figure 2.11 présente la nouvelle courbe de perte pour la reconnaissance des renards.

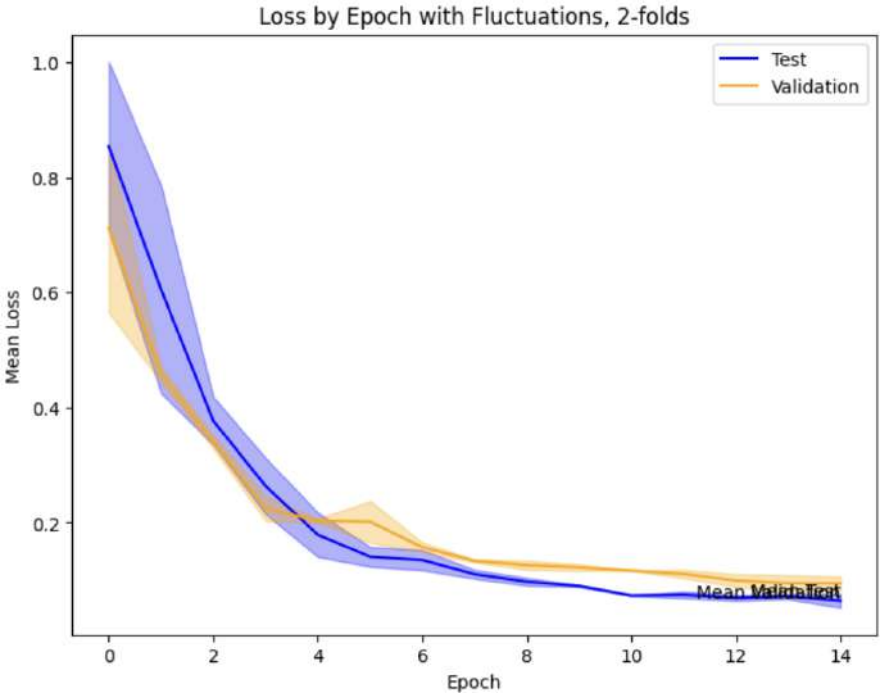

FIGURE 2.11 – Courbe de perte du modèle utilisant ResNet50

Le transfer learning avec ResNet50 a notablement amélioré la reconnaissance des renards, des éléphants et des tigres, avec des taux de précision supérieurs. L'utilisation des connaissances préalables de ResNet50 a permis de réduire le surapprentissage, assurant un modèle plus robuste. Cette approche illustre non seulement l'efficacité du transfer learning sur des jeux de données limités, mais contribue également à réduire l'empreinte environnementale en évitant l'entraînement de nouveaux modèles complexes nécessitant plusieurs unités de calcul.

2.3 Conclusion sur la classification

Au terme de cette exploration dans le domaine de la classification d'images, nous pouvons conclure sur une note d'optimisme quant aux progrès réalisés et aux enseignements tirés. En déployant une série de techniques telles que la normalisation des données, le dropout, et en utilisant le transfer learning avec ResNet50, nous avons constaté une nette amélioration des performances de nos modèles. Nous avons appris à mieux appréhender les défis liés au surapprentissage et à l'hétérogénéité des données, en trouvant des solutions qui ont permis d'atteindre des niveaux de précision plus élevés pour la classification des renards, des éléphants et des tigres.

De plus, cette expérience nous a également sensibilisé à l'importance de l'équilibre entre la qualité des données, la complexité du modèle et la capacité de généralisation. En nous appuyant sur des outils tels que Matplotlib et TensorFlow, nous avons pu visualiser et analyser les performances de nos modèles, identifiant ainsi les points de convergence et les opportunités d'amélioration.

Avec ces connaissances fraîchement acquises, nous sommes prêts à relever de nouveaux défis et à explorer de nouvelles frontières dans ce domaine passionnant. Bien que nous ayons mis en place le transfert d'apprentissage et réduit de manière significative le surapprentissage, nous reconnaissons que notre modèle n'est pas encore parfait en raison du manque de données. Alors que nous clôturons cette étape de classification d'images avec succès, notre voyage dans le monde de l'apprentissage automatique ne fait que commencer. Dans le prochain chapitre, nous aborderons la génération d'images, tout en envisageant de produire des images supplémentaires pour enrichir notre jeu de données.

Chapitre 3

Génération d'images

Dans le chapitre précédent, nous avons abordé le problème du surapprentissage des modèles et exploré diverses solutions, notamment l'augmentation du volume de données disponibles. Le présent chapitre s'inscrit dans cette perspective, avec un focus particulier sur la génération d'images.

Notre objectif principal est de créer de nouvelles images pour enrichir notre ensemble de données, particulièrement axé sur les représentations visuelles des animaux de la classe "renards". Pour ce faire, nous explorons l'utilisation des autoencodeurs, des autoencodeurs variationnels et des Réseaux Antagonistes Génératifs (GAN).

Nous débutons par une revue succincte des autoencodeurs, soulignant leur rôle dans la compression et la reconstruction d'images. Ensuite, nous nous penchons sur les autoencodeurs variationnels, mettant en évidence leur capacité à capturer la distribution probabiliste des données dans un espace latent. Cette exploration nous amène à la section suivante, où nous discutons des résultats obtenus avec les autoencodeurs variationnels et identifions les défis liés à la génération d'images dans l'espace latent.

Enfin, nous examinons le modèle GAN que nous avons développé, illustrant les images générées et évaluant sa performance par rapport aux autres approches explorées.

3.1 Auto-Encodeurs

Dans cette section dédiée aux auto-encodeurs, nous explorons une classe de modèles fondamentaux en apprentissage non supervisé. Nous abordons leur fonctionnement et leur utilité dans la compression et la reconstruction d'images, ainsi que leur rôle dans la génération d'images synthétiques.

3.1.1 Comment ça marche ?

Un autoencodeur est un type particulier de réseau de neurones qui vise à reproduire fidèlement en sortie les données d'entrée, tout en passant par une représentation intermédiaire, appelée "latent space", de dimension réduite. Il se compose de deux parties principales : l'*encodeur* et le *décodeur*.

L'encodeur fonctionne comme un compresseur d'image. Il prend en entrée une image, généralement de grande taille, et la transforme progressivement en une représentation plus compacte. Par exemple, une image de 128x128 pixels pourrait être réduite à une taille de 32x32 pixels. Cette réduction de dimensionnalité est réalisée grâce à des couches de convolution successives, suivies éventuellement de couches de pooling, qui permettent de réduire progressivement la résolution spatiale de l'image tout en préservant ses caractéristiques essentielles.

Une fois l'image compressée, elle est représentée dans ce qu'on appelle le "*latent space*". Cette zone représente un espace de représentation de plus basse dimension dans lequel les caractéristiques les plus importantes de l'image sont encapsulées. C'est dans cet espace latent que le modèle tente de capturer les informations les plus significatives de l'image.

Enfin, le décodeur prend cette représentation réduite de l'image et la reconstruit progressivement pour retrouver l'image originale. Il utilise des techniques d'*upsampling*, telles que les couches de transposition de convolution, pour augmenter progressivement la résolution spatiale de l'image jusqu'à ce qu'elle atteigne sa taille d'origine. L'objectif est de reconstruire une image qui soit la plus fidèle possible à l'image initiale, en utilisant uniquement les informations contenues dans l'espace latent.

La Figure 3.1 montre une représentation de la compression effectuée par l'encodeur de l'image récupérée en entrée, puis la reconstruction de l'image par le décodeur en partent de l'espace latent.

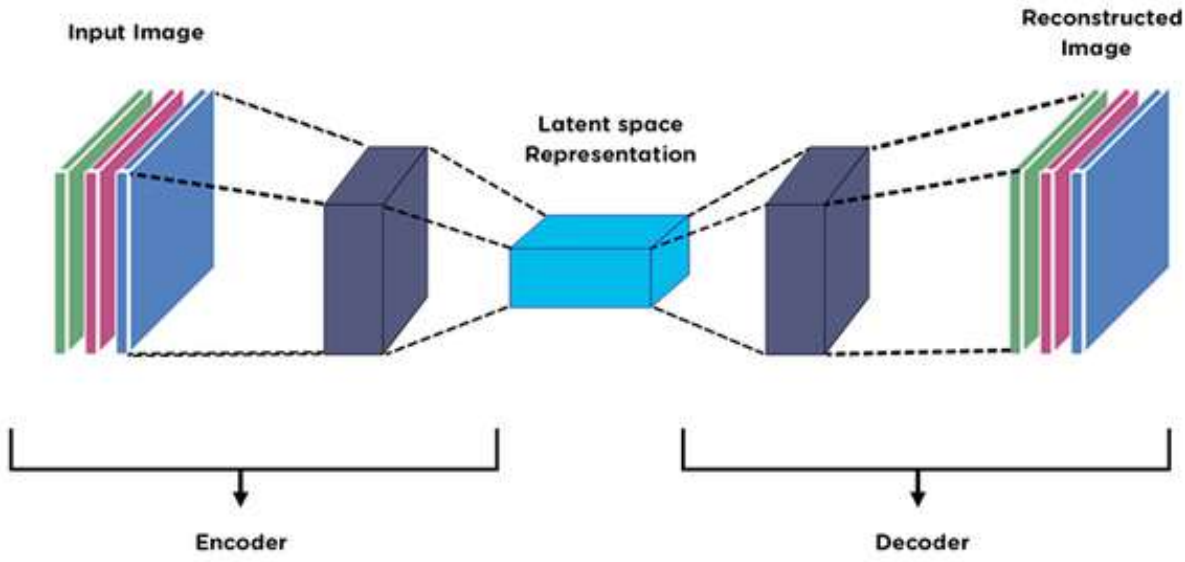

FIGURE 3.1 – Fonctionnement d'un auto-encodeur

3.1.2 En pratique

Cette section explore plusieurs applications des autoencodeurs, mettant en lumière leur utilisation dans la génération d'images, leur capacité à débruiter des images et même à les coloriser. En effet, les autoencodeurs offrent une polyvalence remarquable, permettant de répondre à divers besoins dans le domaine du traitement d'images.

Nous explorons l'utilisation d'un autoencodeur dans le contexte de l'analyse d'une collection d'images mettant en scène des renards dans la neige. Nous évaluons la performance de notre modèle en surveillant son exactitude (accuracy), déterminée par la comparaison entre les caractéristiques de l'image reconstruite en sortie et celles de l'image d'origine en entrée.

Pour cela, nous avons commencé par faire un auto-encodeur de base avec 4 simples couches de convolution, mais nous avons constaté que les résultats n'étaient pas assez satisfaisants. Afin d'améliorer les performances de notre modèle initial, nous avons intégré la *Batch Normalization*. Cette technique régularise les activations des couches cachées, stabilisant ainsi la

distribution des valeurs d'activation et favorisant la convergence de l'entraînement avec des taux d'apprentissage plus élevés.

Génération d'images fidèles à l'origine

Regardons de plus près l'évolution de la capacité de notre modèle à générer des images fidèles aux originales en fonction du nombre d'epochs, reflétée par la diminution de la valeur de la perte (Loss). La Figure 3.2 compare les images d'entrée avec celles générées après **100 epochs**, présentant une perte de 0.014. La Figure 3.3 quant à elle, met en avant les images produites après **1500 epochs** avec une loss de 0.004, témoignant d'une nette amélioration de la netteté.

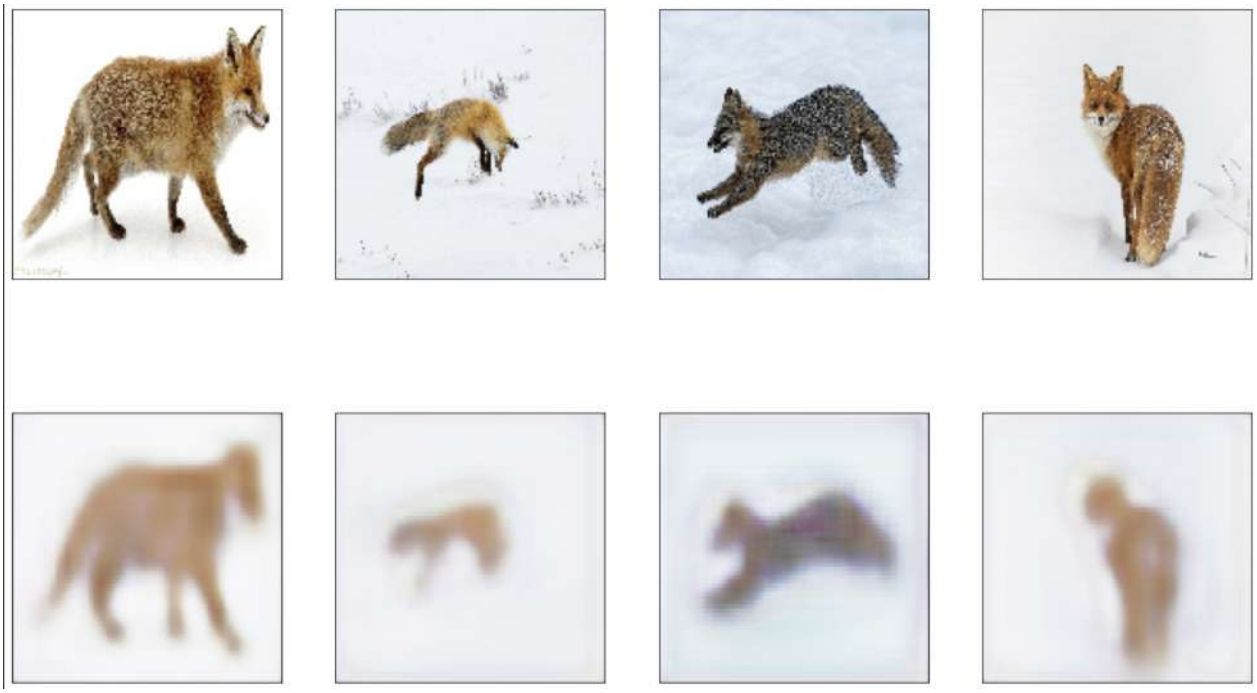

FIGURE 3.2 – Images générées avec 100 epochs

FIGURE 3.3 – Images générées avec 1500 epochs

Avec ImageDataGenerator

Pour atténuer le surapprentissage, nous avons recours à l'augmentation de données à l'aide de ImageDataGenerator, comme décrit dans la Section 2.2.3. La Figure 3.4 illustre que les

images générées présentent une qualité légèrement inférieure à celles obtenues précédemment, mais demeurent globalement satisfaisantes, conservant bien les caractéristiques distinctives du renard ainsi que les nuances de couleurs.

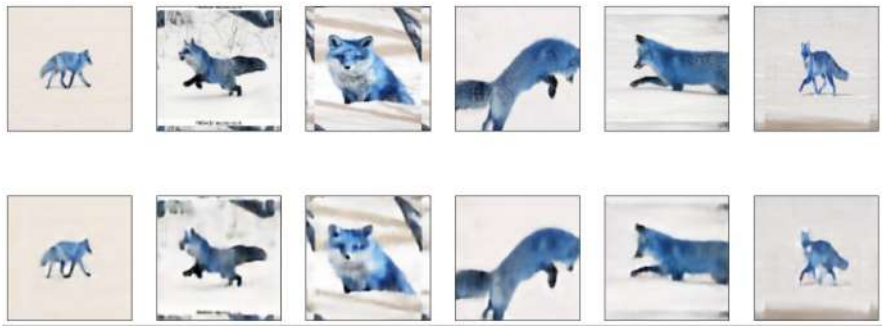

FIGURE 3.4 – Images générées après ImgDataGenerator

Sur des images bruitées

Dans cette étude, notre modèle est appliqué à des images contaminées par du bruit, avec pour objectif de les nettoyer afin de faciliter la distinction des caractéristiques de forme et de couleur des renards. En éliminant le bruit des images, le modèle s'efforce d'améliorer la clarté visuelle et la précision des détails, ce qui devrait permettre une meilleure interprétation et analyse des images par la suite.

La Figure 3.5 ci-dessous met en évidence la disparité entre les images affectées par le bruit, telles qu'elles sont fournies en entrée, et les images produites par l'autoencodeur après le processus de débruitage.

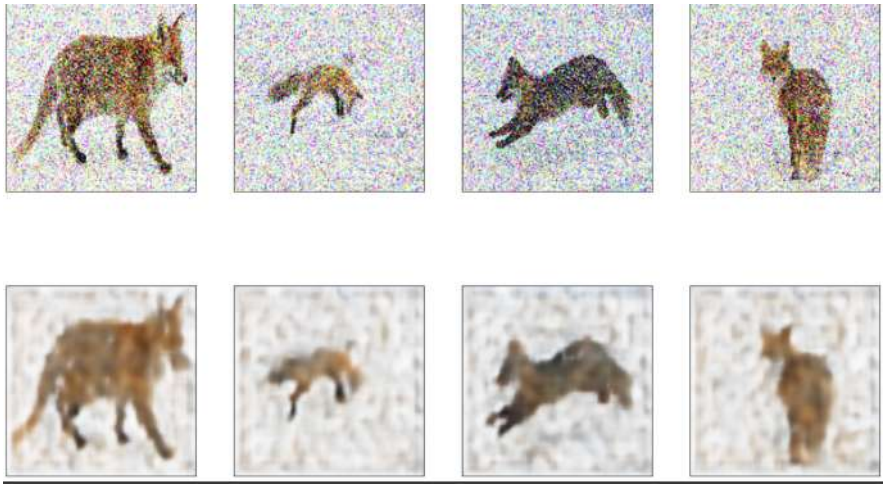

FIGURE 3.5 – Débruiter des images bruitées

Coloriser des images en noir et blanc

Nous procédons à l'entraînement de notre modèle sur des images en noir et blanc afin de lui permettre de les coloriser. Les figures présentent, respectivement, l'image en entrée, la sortie générée par le modèle, et l'image cible attendue. Dans la Figure 3.6, nous comparons les images d'entrée avec celles générées après **1 epoch**, où le modèle parvient déjà à distinguer la forme du renard du fond. Ensuite, dans la Figure 3.7, nous observons les images produites après **150 epochs**, montrant une colorisation assez fidèle à l'originale, bien que légèrement floue, permettant néanmoins d'identifier clairement un renard dans la neige.

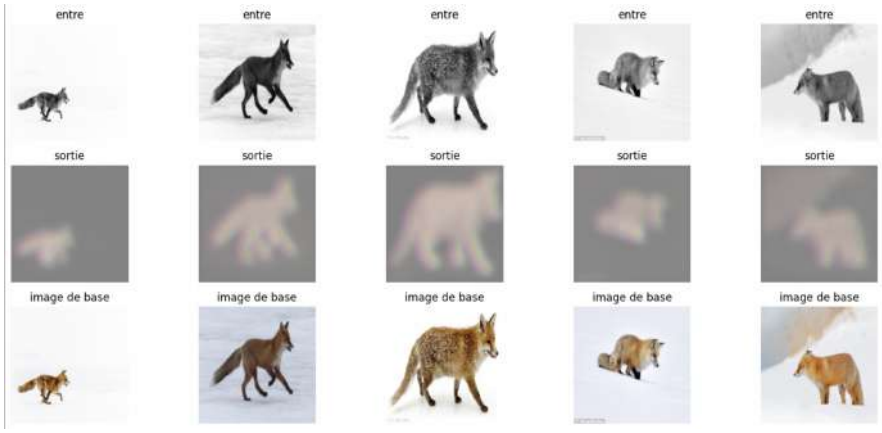

FIGURE 3.6 – Colorisation après une epoch

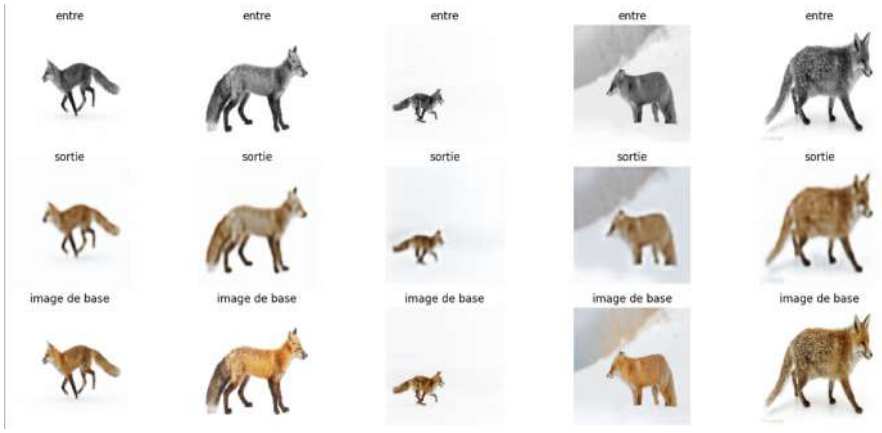

FIGURE 3.7 – Colorisation après 150 epochs

3.1.3 Conclusion sur les Auto-Encodeurs

Les autoencodeurs démontrent leur efficacité dans la restauration d'images à partir de leur version compressée, suggérant leur potentiel pour la génération d'images. Cette observation soulève naturellement l'intérêt d'explorer les autoencodeurs variationnels, qui non seulement reconstruisent les données d'entrée, mais capturent également la distribution probabiliste de celles-ci dans l'espace latent, offrant ainsi une approche plus riche et diversifiée pour la génération d'images.

3.2 Auto-Encodeurs Variationnels (VAE)

Dans cette section, nous explorons les autoencodeurs variationnels (VAE), une extension robuste des autoencodeurs conventionnels, qui introduisent une approche probabiliste pour représenter l'espace latent. En modélisant la distribution probabiliste des données dans cet espace, les VAE offrent une approche plus riche et variée pour la génération d'images et la modélisation des données.

3.2.1 L'espace latent

Revenons tout d'abord sur la notion de *latent space* que nous avons eu le temps de survoler assez rapidement dans la Section 3.1.1. Le latent space représente une représentation compressée des caractéristiques des images comme expliqué précédemment. Après avoir été traitées par l'encodeur, les images sont transformées en vecteurs de données de dimension réduite dans cet espace latent.

Chaque point dans l'espace latent correspond à une représentation unique d'une image. Les similitudes entre les images sont reflétées par la proximité de leurs représentations dans cet espace. Par exemple, des images similaires, comme celles présentant des visages avec des expressions faciales similaires, tendent à être regroupées dans des régions voisines de l'espace latent.

L'espace latent constitue donc une représentation compressée et abstraite des images, où les informations essentielles sont encapsulées pour faciliter la reconstruction précise des images lors du décodage. Bien que crucial pour la génération de nouvelles images, les autoencodeurs traditionnels présentent une faiblesse à ce niveau.

En effet, ils peuvent ne pas gérer de manière optimale la distribution des images dans l'espace latent, ce qui peut conduire à des regroupements incohérents. Par exemple, des images similaires pourraient être dispersées dans des zones distantes, tandis que des images différentes pourraient se retrouver dans la même zone. Les autoencodeurs variationnels offrent une solution à ce problème en contrôlant la distribution de l'espace latent à l'aide de calculs de probabilité, assurant ainsi une représentation plus structurée et cohérente des données.

3.2.2 Comment ça marche ?

Un autoencodeur variationnel (VAE) vise à "forcer" l'espace latent à avoir une distribution normale, ce qui permet une représentation structurée des données. Pour ce faire, le VAE calcule un vecteur moyen et un écart type dans l'espace latent, ce qui est réalisé en ajoutant deux couches (dense) au modèle après les convolutions pour obtenir ces paramètres. Ensuite, une fonction d'échantillonnage est utilisée pour générer un échantillon de la distribution à partir du vecteur moyen et de l'écart type.

Pour évaluer la qualité de la distribution dans l'espace latent, une divergence de Kullback-Leibler est calculée pendant la backward propagation : Si l'échantillon proposé n'est pas conforme à la distribution normale, la divergence de Kullback-Leibler (KL) sera plus élevée, on cherche donc à minimiser cette valeur. Cela permet de guider l'apprentissage pour s'assurer que l'espace latent suit une distribution souhaitée.

La Figure 3.8 présente un schéma simplifié illustrant la gestion de la distribution des données dans l'espace latent d'un autoencodeur variationnel (VAE). Les vecteurs "moyenne" et "écart type", situés entre l'encodeur et le décodeur, représentent les paramètres de distribution qui sont utilisés pour échantillonner des points dans l'espace latent. Ces échantillons sont ensuite évalués à la sortie du décodeur en utilisant une fonction d'erreur, à laquelle est

ajoutée la divergence de Kullback-Leibler (KL).

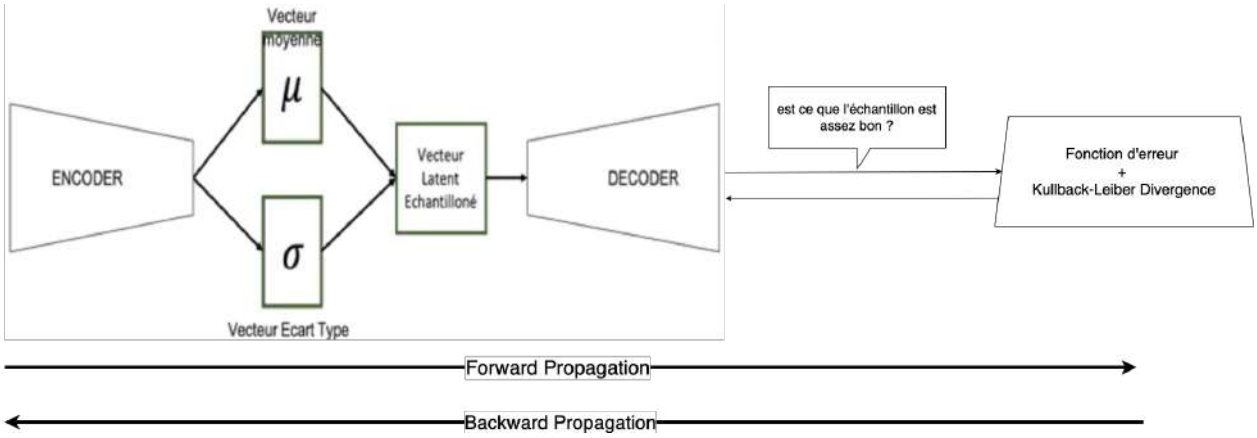

FIGURE 3.8 – Schéma de base d'un VAE

3.2.3 En pratique

Nous entamons désormais la phase de génération de nouvelles images, en explorant tout d'abord la répartition des données dans l'espace latent de notre autoencodeur variationnel (VAE). Cette étape nous permettra de comprendre comment les caractéristiques des images sont encapsulées et organisées dans cet espace latent. Par la suite, nous observerons avec intérêt les images de renards générées par notre VAE, afin d'évaluer sa capacité à produire des représentations fidèles et diversifiées de cette classe d'images.

Analysons les performances de notre VAE

Nous avons utilisé une fonction de visualisation permettant d'analyser la représentation des images dans l'espace latent. Cette fonction, nommée "visualize", prend en entrée une image et utilise l'encodeur du VAE pour obtenir sa représentation dans l'espace latent. Ensuite, elle affiche à la fois l'image originale et sa représentation dans une figure, offrant ainsi une comparaison visuelle entre les deux.

On peut remarquer d'après la Figure 3.9 ci-dessous une homogénéité dans la distribution des images dans l'espace latent. Cela se traduit par une organisation cohérente des caractéristiques des images, où des images similaires sont regroupées ensemble dans des zones spécifiques de l'espace latent. Cette homogénéité est essentielle pour la génération d'images, car elle indique que le VAE parvient à capturer efficacement les structures sous-jacentes des données et à les représenter de manière cohérente dans l'espace latent.

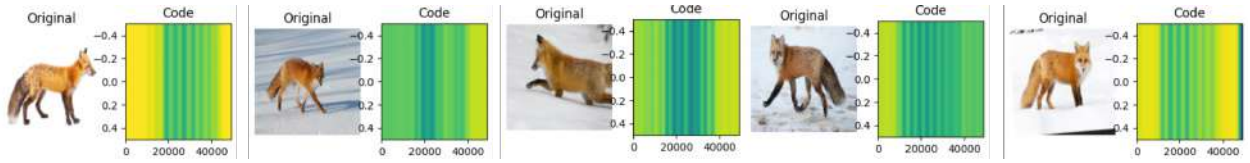

FIGURE 3.9 – Visualisation de la répartition des images dans l'espace latent

Regardons notre espace latent

La Figure 3.10 illustre l'espace latent généré par notre VAE. Pour une meilleure appréhension de la distribution des images dans cet espace, un zoom est effectué, comme présenté dans la Figure 3.11. Cette visualisation met en évidence une organisation cohérente des images similaires, confirmant ainsi la capacité du VAE à regrouper efficacement les caractéristiques visuelles des données. Ces observations prometteuses nous incitent à exploiter pleinement cet espace latent pour la génération de nouvelles images.

FIGURE 3.10 – Espace latent

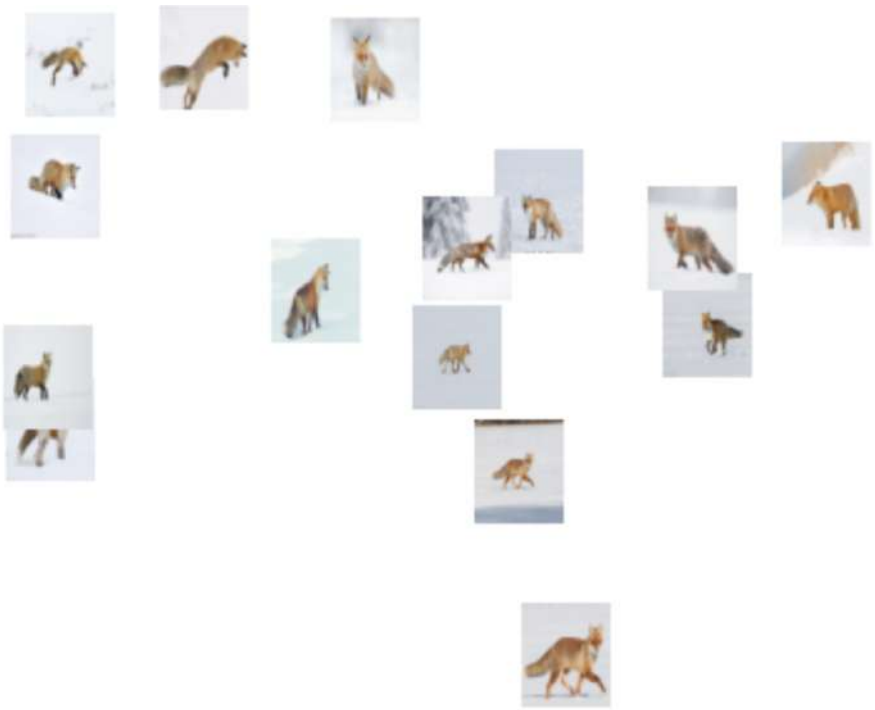

FIGURE 3.11 – Zoom sur l'espace latent

Générons des images

Il nous reste désormais à relever le défi de rivaliser avec les plus grands générateurs d'images en produisant nos propres créations. La figure 3.12 présente les résultats obtenus après environ 6000 epochs et de nombreuses heures de calcul intensif. Bien que nous soyons encore loin de pouvoir concurrencer les leaders du domaine, nous pouvons déjà distinguer la forme et la couleur d'un renard (ou d'un singe mutant..) dans la neige, ce qui constitue un accomplissement significatif. Toutefois, il est important de noter que des améliorations supplémentaires pourraient être réalisées avec un plus grand volume de données, un nombre d'epochs accru et une architecture plus complexe. Pour l'instant, nous nous satisferons de ces résultats, tenant compte du temps, des ressources nécessaires, et surtout de l'impact environnemental associé.

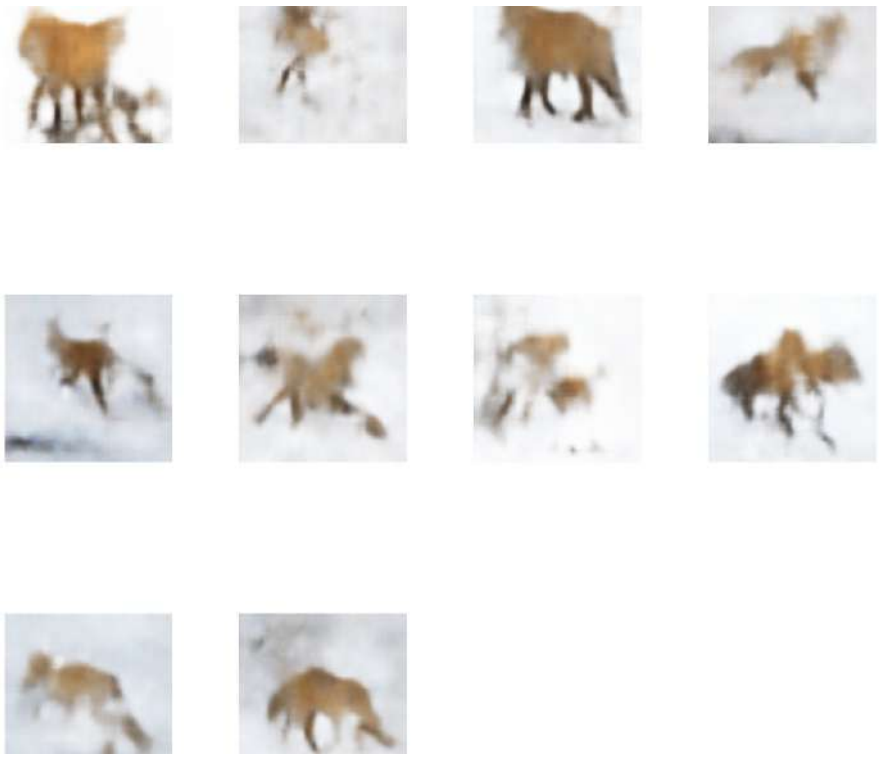

FIGURE 3.12 – Images générées grâce au latent space

3.2.4 Conclusion sur les VAE

Les autoencodeurs variationnels (VAE) se révèlent être des outils puissants pour la génération d'images, offrant une approche probabiliste pour la modélisation de l'espace latent. Malgré leurs succès dans la compression et la reconstruction d'images, les VAE présentent encore des défis en ce qui concerne la génération d'images diversifiées et fidèles. Une ouverture prometteuse pour relever ces défis réside dans l'exploration et l'intégration des Réseaux Antagonistes Génératifs (GAN), qui offrent une approche alternative et complémentaire en matière de génération d'images.

3.3 Réseaux Antagonistes Génératifs

Les Réseaux Génératifs Antagonistes (GANs) ont révolutionné le domaine de l'intelligence artificielle en permettant la génération de données réalistes à partir de zéro. Leur fonctionnement repose sur un processus d'entraînement "adversatif" entre un *générateur*, agissant tel un décodeur, et un *discriminateur*, qui joue le rôle d'un classifieur. Cette section examinera en détail l'architecture, le processus d'entraînement et les capacités de génération d'images offertes par les GANs.

3.3.1 Architecture

Dans l'architecture des Réseaux Génératifs Antagonistes (GANs), deux réseaux neuronaux interagissent de manière antagoniste : le générateur et le discriminateur. Le générateur est chargé de créer des images réalistes à partir d'un bruit latent, initialement distribué de manière aléatoire. Il utilise des opérations de convolution pour transformer ce bruit en une représentation d'image. En parallèle, le discriminateur prend en entrée des images générées par le générateur ainsi que des images réelles, et tente de les classifier comme réelles ou fausses. L'erreur résultant de cette classification est renvoyée au générateur, l'incitant à améliorer ses poids pour produire des images de meilleure qualité. De son côté, le discriminateur est un classifieur classique, avec des couches de convolution et une fonction d'activation de type sigmoïde en sortie pour distinguer les images réelles des images générées. La mise à jour des poids du discriminateur se fait dans le but de mieux classifier les images produites par le générateur, tandis que le générateur ajuste ses poids pour tromper le discriminateur en générant des images de plus en plus réalistes. Ainsi, cette architecture fonctionne de manière itérative, chaque réseau cherchant à améliorer sa performance tout en tentant de surpasser l'autre.

Cependant une question clé se pose : comment le discriminateur met-il à jour ses poids si le résultat de sa fonction d'activation est destiné au générateur ? Cette interrogation initiale ouvre la voie à une exploration plus approfondie du processus d'entraînement d'un GAN, mettant en lumière la dynamique complexe qui sous-tend l'apprentissage antagoniste du générateur et du discriminateur.

La Figure ci-dessous illustre comment le générateur crée des données synthétiques à partir d'un bruit aléatoire, qui seront jugées par le discriminateur comme vraies ou fausses, puis nous renvoyons la décision du discriminateur au générateur. En s'entraînant de manière antagoniste, le générateur cherche à améliorer la qualité de ses données pour tromper le discriminateur, créant ainsi des données de plus en plus réalistes.

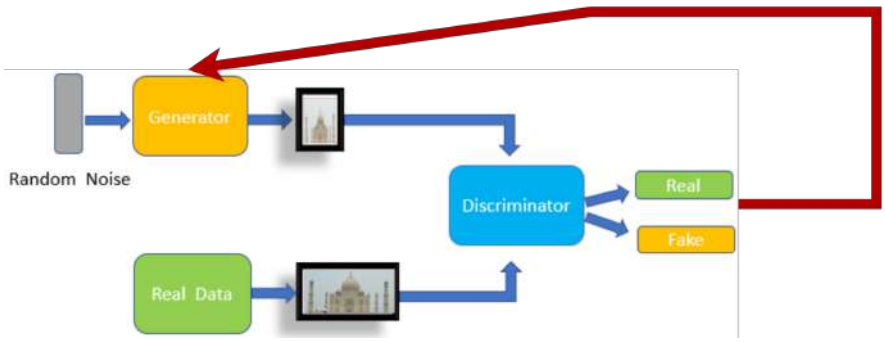

FIGURE 3.13 – Architecture d'un GAN

3.3.2 Entraînement d'un GAN

Le processus d'entraînement d'un GAN est un ballet subtil entre le générateur et le discriminateur dans un cadre d'apprentissage antagoniste.

Lors de la première étape, le discriminateur est initié en lui fournissant un mélange d'images réelles provenant de l'ensemble de données réel et d'images générées par le générateur. En utilisant la méthode train_on_batch, qui permet d'entraîner un modèle en lui fournissant progressivement des petits groupes de données (batches), le discriminateur est entraîné à distinguer les vraies images des fausses en minimisant sa valeur de perte (loss). Cette étape vise à améliorer la capacité du discriminateur à classifier correctement les données.

Dans la deuxième étape, le générateur entre en jeu. Son objectif est de produire des données qui ressemblent autant que possible à des données réelles pour tromper le discriminateur. Cependant, contrairement au discriminateur, le générateur n'est pas directement entraîné par la méthode train_on_batch. Au lieu de cela, le générateur est intégré dans le modèle GAN complet, où il est encouragé à optimiser les données générées pour maximiser la probabilité que le discriminateur se trompe en les classant comme réelles. Cette optimisation est réalisée à travers l'entraînement global du modèle GAN, où les poids du générateur sont ajustés pour améliorer sa capacité à générer des données réalistes.

De cette manière, bien que la fonction train_on_batch soit principalement utilisée pour entraîner le discriminateur, le générateur est entraîné de manière indirecte à travers le processus global d'entraînement du modèle GAN. Cette approche permet une dynamique compétitive et interactive entre les deux réseaux, conduisant éventuellement à la génération de données de haute qualité.

3.3.3 En pratique

En réalité, cette compétition entre le générateur et le discriminateur peut s'avérer plus complexe et exiger davantage de temps et de ressources que d'autres méthodes d'apprentissage automatique, telles que les autoencodeurs variationnels (VAE) que nous avons étudiés dans la Section 3.2. Les GANs peuvent nécessiter un temps de convergence plus long en raison de leur processus d'entraînement adversatif, qui implique une lutte constante entre le générateur et le discriminateur.

En ce qui concerne la qualité des résultats, les GANs ont tendance à mieux performer sur des ensembles de données comportant un grand nombre d'images, contrairement aux VAE. Ainsi, nous avons augmenté notre ensemble de données en doublant le nombre d'images, tout en surveillant attentivement l'évolution des deux réseaux de neurones. Il est crucial de maintenir un équilibre dans l'apprentissage des deux réseaux, car si l'un apprend plus rapidement que l'autre, cela peut entraîner un déséquilibre et une grande déception en sortie au bout de plusieurs heures. Par conséquent, l'augmentation de la taille de l'ensemble de données est essentielle pour assurer un apprentissage équilibré des GANs.

En ce qui concerne notre capacité à rivaliser avec OpenAI, nous procéderons à une évaluation à différentes étapes de l'entraînement de nos modèles. Nous examinerons attentivement les résultats après 50 epochs dans la Figure 3.14, puis après 1000 epochs dans la Figure 3.15. Enfin, nous nous arrêterons à 7000 epochs, comme indiqué dans la Figure 3.16, pour évaluer l'évolution de la qualité des images générées. Ces illustrations mettront en évidence les progrès réalisés, notamment en termes de couleurs plus réalistes et de formes plus distinctes, permettant une meilleure identification des détails d'un renard dans la neige, par rapport aux images obtenues par les VAE (voir Figure 3.12).

FIGURE 3.14 – Images générées avec 50 epochs

FIGURE 3.15 – Images générées avec 1000 epochs

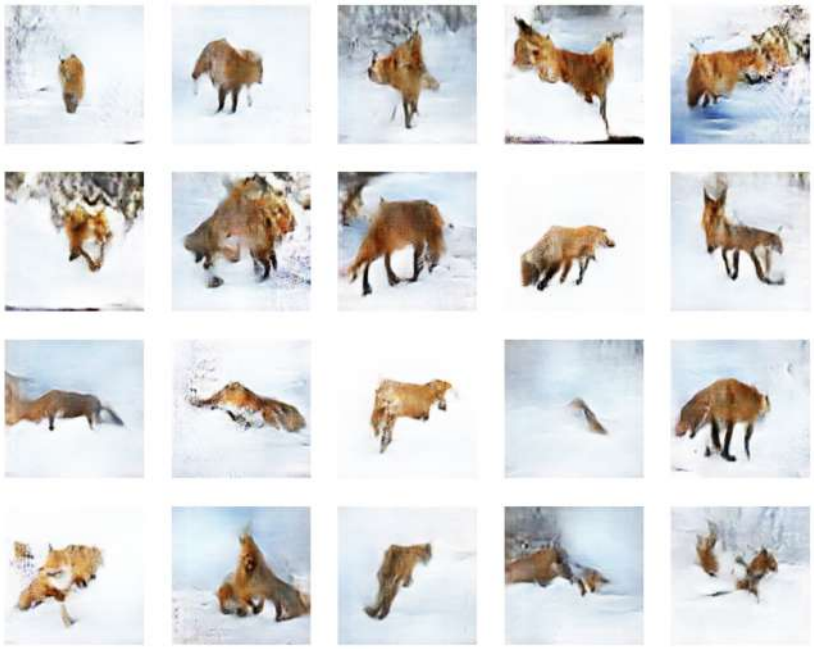

FIGURE 3.16 – Images générées avec 7000 epochs

3.3.4 Conclusion sur les GANs

Dans notre exploration des Réseaux Génératifs Antagonistes (GANs), nous avons plongé dans un univers complexe où la rivalité entre le générateur et le discriminateur engendre des résultats captivants. Tout au long de l'entraînement progressif de nos modèles, nous avons constaté une nette amélioration dans la qualité des images générées, démontrant ainsi le potentiel considérable des GANs dans la production de données réalistes. Cependant, cette approche représente simplement l'une des nombreuses méthodes disponibles dans le vaste domaine de l'apprentissage automatique. Des leaders comme OpenAI explorent également des approches novatrices, bien que fondamentalement similaires à celles détaillées ici, telles que l'introduction de bruit dans des images réelles pour stimuler la créativité du modèle [3].

3.4 Conclusion sur la Génération d'images

Dans ce chapitre, nous avons exploré deux approches majeures de génération de données en apprentissage automatique : les Autoencodeurs Variationnels (VAE) et les Réseaux Génératifs Antagonistes (GANs). Chacune de ces méthodes présente des avantages distincts ainsi que des limites à prendre en considération.

Les VAE offrent une approche probabiliste pour générer des données, en permettant de modéliser explicitement la distribution des données latentes. Leur capacité à générer des données réalistes tout en préservant la structure latente des données en fait une méthode attrayante pour diverses applications. De plus, les VAE offrent un cadre clair pour l'interprétation des données générées, ce qui peut être crucial dans certains domaines.

Cependant, les VAE peuvent souffrir d'un problème de qualité de reconstruction, produisant parfois des images floues ou déformées. De plus, la génération de données avec les VAE peut être moins diversifiée et moins précise que celle des GANs, en raison de la contrainte supplémentaire imposée par la distribution normale de la couche latente.

D'autre part, les GANs ont révolutionné la génération de données en introduisant un cadre d'apprentissage antagoniste entre un générateur et un discriminateur. Cette approche conduit souvent à des résultats visuellement plus impressionnants, avec des images plus nettes et plus réalistes. De plus, les GANs sont connus pour leur capacité à générer des données hautement diversifiées et à capturer des structures complexes dans les données.

Cependant, les GANs peuvent être plus difficiles à entraîner et à stabiliser en raison de la compétition entre le générateur et le discriminateur, ce qui nécessite souvent plus de temps et de ressources. De plus, les GANs peuvent souffrir du problème de mode collapse, où le générateur produit des données de manière répétitive ou peu variée.

En ce qui concerne notre rivalité avec les leaders du domaine de la génération d'images, il est clair que nous progressons dans la bonne direction. Cependant, ces résultats ont déjà eu un certain coût financier et ont entraîné une empreinte carbone significative. Pour rivaliser pleinement avec les géants de la génération d'images, quelques millions d'euros supplémentaires devront être investis. De plus, nous aurons besoin de ressources humaines considérables, notamment des centaines d'employés motivés pour trier nos données d'apprentissage et une infrastructure informatique robuste basée dans des zones à basse température pour réduire les coûts énergétiques.

Chapitre 4

Organisation

Le processus de gestion adopté pour notre projet d'apprentissage automatique s'est avéré méthodique et hautement collaboratif, en cohérence avec les principes de la méthode agile. Nous avons organisé des réunions formelles avec notre encadrant, Monsieur Pascal Poncelet, où nous fixions les objectifs à atteindre avant la prochaine réunion et abordions de nouveaux concepts toutes les 2 à 3 semaines. Ces échanges réguliers nous ont permis d'évaluer les progrès réalisés, de discuter des tâches à venir et de résoudre rapidement les problèmes éventuels, maintenant ainsi une vision claire de l'avancement du projet et garantissant une prise de décision rapide et éclairée.

En dehors des réunions formelles, nous avons utilisé un groupe Discord pour faciliter la communication et la collaboration entre les membres de l'équipe et Monsieur Poncelet, favorisant ainsi des échanges fluides et une résolution rapide des problèmes.

Notre approche de développement a été itérative et incrémentielle, mettant l'accent sur la livraison continue de fonctionnalités. Bien que les besoins du projet n'aient pas été sujets à des changements majeurs, nous avons maintenu une flexibilité suffisante pour nous adapter aux ajustements nécessaires en cours de route. Chaque itération du projet a apporté son lot d'améliorations et de raffinements, renforçant ainsi la qualité globale du travail accompli.

La rédaction du rapport s'est déroulée de manière progressive et alignée sur l'évolution du projet, assurant ainsi une documentation constamment actualisée de nos travaux. Cette approche nous a permis de capturer de manière exhaustive les détails et les nuances du projet, facilitant ainsi la transition entre les différentes phases de développement.

En résumé, notre projet d'apprentissage automatique a été caractérisé par une gestion rigoureuse, une communication transparente et une collaboration efficace, des facteurs qui ont grandement contribué à son succès.

Chapitre 5

Conclusion

Ce rapport marque la fin d'un périple captivant à travers les méandres de l'apprentissage automatique, abordant successivement la classification d'images, la modélisation probabiliste avec les Variational Autoencoders (VAEs), et l'exploration des territoires innovants des Réseaux Génératifs Antagonistes (GANs).

Dans la première partie de notre travail, nous avons plongé dans l'univers des CNN pour la classification d'images. En déployant des architectures sophistiquées et des méthodes d'entraînement avancées, nous avons atteint des niveaux de précision impressionnants dans la catégorisation d'images, soulignant ainsi l'efficacité et la polyvalence de cette approche dans la résolution de problèmes complexes.

En poursuivant notre exploration, nous avons abordé les VAEs, une technique novatrice qui allie la puissance des réseaux de neurones avec des concepts de probabilité. Bien que les VAEs ne produisent pas toujours des images aussi saisissantes que les GANs, leur capacité à modéliser des distributions latentes en fait un outil précieux pour la génération de données complexes et la compréhension des structures sous-jacentes des données.

Enfin, notre périple nous a conduit à plonger dans les profondeurs intrigantes des GANs, où la rivalité entre le générateur et le discriminateur a donné naissance à des résultats étonnants dans la génération d'images réalistes. Cette exploration a révélé le potentiel immense des GANs pour la création de données visuelles authentiques et captivantes, ouvrant la voie à de nouvelles applications dans des domaines tels que la création artistique.

Au-delà des résultats obtenus, ce projet a été une leçon précieuse sur les processus de collaboration, de gestion de projet et d'apprentissage continu. Les réunions régulières, la communication transparente et la rédaction progressive du rapport ont été des piliers essentiels de notre succès, illustrant ainsi l'importance de la méthodologie et de la collaboration dans les projets complexes d'apprentissage automatique.

En définitive, ce rapport témoigne de notre engagement envers l'excellence, de notre passion pour l'innovation et de notre détermination à repousser les limites de l'intelligence artificielle. Alors que ce projet prend fin, il laisse derrière lui un héritage d'apprentissage, de croissance et d'inspiration pour les futurs travaux dans ce domaine passionnant de l'informatique.

Nous tenons également à exprimer notre sincère gratitude envers notre encadrant, Monsieur Pascal Poncelet, pour son soutien constant, ses conseils précieux et son expertise inestimable tout au long de ce projet. Sa contribution a été essentielle à notre réussite et nous lui en sommes profondément reconnaissants.

En espérant que votre découverte de ce domaine ait été aussi enrichissante et passionnante qu'elle l'a été pour nous, nous tenons à vous remercier d'avoir parcouru ce rapport jusqu'à son terme. Votre intérêt et votre engagement envers ce sujet complexe sont une source d'inspiration pour notre équipe, et nous espérons que les connaissances partagées ici vous seront

aussi précieuses qu'elles l'ont été pour nous. Merci de votre attention et de votre soutien tout au long de ce voyage à travers l'apprentissage automatique.

Glossaire

Epoch :

Une epoch est une seule itération complète de l'ensemble des données d'entraînement par un modèle.

Batch size :

La taille du lot (ou batch size) est le nombre d'exemples d'entraînement utilisés dans une seule mise à jour des poids du modèle.

Learning rate :

Le taux d'apprentissage est un paramètre qui contrôle la taille des pas de mise à jour des poids du modèle lors de l'entraînement.

k-folds :

La validation croisée k-folds divise l'ensemble de données en k sous-ensembles pour évaluer la performance du modèle.

Loss :

La perte est une mesure de l'erreur du modèle par rapport aux valeurs attendues lors de l'apprentissage.

Accuracy :

L'exactitude est une mesure de performance indiquant le pourcentage de prédictions correctes du modèle sur l'ensemble des données de test.

Table des figures

Bibliographie

[1] URL : https://colab.research.google.com.

[2] URL : https://scikit-learn.org/stable/auto_examples/index.html#model-selection.

[3] URL : https://neptune.ai/blog/6-gan-architectures.

[4] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.