

UNIVERSITÉ MOHAMMED V DE RABAT
Faculté des Sciences



Département d’Informatique
Filière Licence Fondamentale
en Sciences Mathématiques et Informatique

PROJET DE FIN D’ÉTUDES

Analyse comparative de la classification d’images
avec les CNN et GNN

Présenté par :
ALAOUI ISMAILI MARYAM
EL HAMDI SALMA

soutenu le 09 Juin 2023 devant le Jury

Mr A.Drissi ELMaliani Professeur à la Faculté des Sciences - Rabat *Encadrant*
Mme Dounia Lotfi Professeur à la Faculté des Sciences - Rabat *Examinaterice*

Année Universitaire 2022-2023

Remerciements

En préambule à ce mémoire on remercie Dieu qui nous a aidés et nous a donné la patience et le courage durant ces longues années d'étude.

Aussi nos remerciements au corps le directeur de ce mémoire , monsieur Ahmed DRISSI EL MALIANI, pour sa réactivité, sa disponibilité et surtout ses judicieux conseils, qui déploient de grands efforts pour nous assurer une très bonne formation. Un grand merci pour ses conseils concernant mon style d'écriture, ils ont grandement facilité mon travail.

nous remercions sincèrement à tous les professeurs, intervenants ainsi que notre jurys, qui se sont toujours montrés disponibles tout au long de la réalisation de ce mémoire, ainsi pour l'inspiration, l'aide et le temps qu'ils ont bien voulu nous consacrer, et sans eux, ce mémoire n'aurait jamais vu le jour.
Enfin, nos remerciements nos parents, qui ont toujours été là pour nous. nos remerciements nos frères, pour leurs encouragements, nos amis pour Leur soutien inconditionnel et leurs encouragements ont été d'une grande aide.

Résumé

L'objectif de ce projet de fin d'études est de réaliser une étude comparative entre les réseaux neuronaux à graphes (GNN) et les réseaux de neurones convolutionnels (CNN) pour la classification d'images. Le but est de développer des modèles d'apprentissage automatique basés sur les GNN et les CNN respectivement, afin de classer les images dans différentes catégories. Les modèles seront entraînés pour reconnaître les éléments présents dans les images et prédire leur contenu. Par exemple, si une image contient un chat ou un chien, les modèles devront être capables d'identifier et de classer avec précision l'image en conséquence. Une comparaison complète sera réalisée pour évaluer les performances et les caractéristiques des deux approches.

Abstract

The objective of this end-of-studies project is to conduct a comparative study between graph neural networks (GNN) and convolutional neural networks (CNN) for image classification. The goal is to develop machine learning models based on GNN and CNN, respectively, to classify images into different categories. The models will be trained to recognize the elements present in the images and predict the content of each image. For example, if an image contains a cat or a dog, the models should be able to accurately identify and classify the image accordingly. A comprehensive comparison will be conducted to evaluate the performance and characteristics of both approaches.

Table des matières

Remerciements	1
Résumé	2
Abstract	3
Introduction	6
1 Introductions aux Images	7
1.1 Définitions et caractéristiques d'image numérique	7
1.2 Types des images numériques	8
1.2.1 Images binaires	8
1.2.2 Image en niveaux de gris	8
1.2.3 Image couleur (ou RGB)	8
1.3 Formats de fichier et compression d'image	9
1.3.1 BMP (bitmap)	9
1.3.2 GIF (GraphicInterchange Format)	10
1.3.3 JPEG (Joint Photo Expert Group)	10
1.4 Espaces de couleurs	10
1.4.1 Modèles de couleur (RVB, CMJN, etc.)	10
1.5 Applications aux traitements d'image	11
1.5.1 Système de traitement d'images	11
1.5.2 Pré-traitement d'images	11
1.6 Conclusion.	13
2 Intelligence artificielle, Machine learning et Deep learning	14
2.1 Introduction	14
2.2 Intelligence artificielle	15
2.3 Machine learning	15
2.3.1 Les types de machine learning	15
2.4 Deep learning	17
2.5 La relation entre la Machine Learning et Deep Learning	17
2.6 Brève description d'algorithme populaire	18
2.6.1 Neural Networks	18
2.7 Conclusion	19

3 La vision par ordinateur	20
3.1 Introduction à la vision par ordinateur	20
3.2 Les fondements de la vision par ordinateur	20
3.2.1 Système visuel humain et perception visuelle	20
3.2.2 Fondements théoriques de la vision par ordinateur	21
3.3 Classification d'images : Concepts et méthodes	21
3.3.1 Introduction à la classification des images	21
3.3.2 Méthodes de classification des images	22
3.3.3 Entraînement des modèles de classification d'images	22
3.4 Conclusion	23
4 Classification des images à l'aide de CNN et GNN	24
4.1 Réseaux des neurones convolutifs (CNN)	24
4.2 Les réseaux de neurones graphiques (GNN)	26
4.3 Conclusion	28
5 Implementation	29
5.1 Environnement et langages de développement	29
5.1.1 Python	29
5.1.2 Jupyter Notebook	29
5.2 Bibliothèques utilisées	30
5.2.1 PyTorch	30
5.2.2 Matplotlib	31
5.3 Implémentation du modèle CNN	31
5.3.1 Le jeu de donnée utilisé	31
5.3.2 L'architecture du modèle CNN proposé	33
5.3.3 Méthodes d'entraînement	34
5.3.4 Résultats obtenue	35
5.4 Implementation du modèle GNN	38
5.4.1 Le Jeu de donnée utilisée	38
5.4.2 L'architecture du modèle GNN proposé	38
5.4.3 Méthodes d'entraînement	40
5.4.4 Résultats obtenue	41
Conclusion	45

Table des figures

1.1	Représentation d'une image numérique	8
1.2	Représentation d'une image binaire	8
1.3	Représentation image en niveaux de gris	9
1.4	Représentation image couleur	9
1.5	Représentation d'une image en bitmap	10
1.6	modèles de couleur RVB	11
1.7	modèles de couleur CMJN	12
1.8	Schéma d'un système de traitement d'images	12
1.9	Pré-traitement d'images	13
2.1	Intelligence artificielle, Machine learning et Deep learning	14
2.2	Types de Machine Learning	16
2.3	Machine Learning et Deep Learning	18
2.4	Neural networks	18
2.5	Fonctions d'activation	19
3.1	Machine Vision VS Human Vision	21
3.2	Classification des images	22
4.1	La couche de Convolution	25
4.2	La Couche de Pooling	25
4.3	La couche ReLU	26
4.4	La Couche fully-connected	27
4.5	Les réseaux de neurones graphiques	27
4.6	Les Couches de convolution graphique	28
5.1	Python	29
5.2	Jupyter	30
5.3	Pytorch	30
5.4	matplotlib	32
5.5	MNIST dataset	32
5.6	Téléchargement et prétraitement des images	33
5.7	La modèle CNN proposé	34
5.8	Test d'accuracy	35
5.9	Training Loss	36
5.10	Training Accuracy et test Accuracy	36

5.11 Résultat (nombre 9)	37
5.12 Résultat (nombre 4)	37
5.13 CIFAR-10	38
5.14 L'architecture du modèle GNN	39
5.15 L'architecture générale du modèle GNN	40
5.16 Résultats de l'entraînement	42
5.17 Training Loss GNN	42
5.18 test et training accuracy	43
5.19 Résultat (classe "ship")	43
5.20 Résultat (classe "frog")	44

Liste des abréviations

AI Intelligence Artificielle

CIFAR Canadian Institute For Advanced Research

RELU REctified Linear Unit

CNN Convolutional Neural Network

ANN Artificial Neural Network

GPU Graphics Processing Unit

FC Fully-Connected

GNN Graph Neural Network

GCN Graph Convolutional Network

CPU Central Processing Unit

CUDA Compute Unified Device Architecture

MNIST Mixed National Institute of Standards and Technology

Introduction

La classification précise des images est une tâche essentielle dans le domaine de la vision par ordinateur, et de nombreuses approches ont été développées pour atteindre cet objectif. Parmi ces approches, deux architectures de réseaux neuronaux se sont démarquées : les réseaux de neurones convolutifs (CNN) et les réseaux neuronaux à graphes (GNN). Les CNN ont été largement utilisés et ont montré d'excellentes performances dans la classification d'images, tandis que les GNN ont récemment suscité un intérêt croissant en raison de leur capacité à capturer les relations complexes entre les éléments d'une image.

Dans ce rapport de projet de fin d'études, nous nous intéressons à une étude comparative approfondie entre les GNN et les CNN pour la classification des images. L'objectif est de comprendre les forces et les faiblesses de chaque approche et d'évaluer leur performance respective dans des scénarios de classification d'images.

Cette étude comparative nous permettra de mieux comprendre comment les GNN et les CNN abordent la tâche de classification d'images, en mettant l'accent sur leurs différences fondamentales en termes d'architecture, de capacité de représentation et de capacité à capturer les informations contextuelles. Nous examinerons également les avantages et les limitations de chaque approche, ainsi que leurs applications potentielles dans des domaines spécifiques de la vision par ordinateur.

En réalisant cette étude comparative, nous espérons apporter des éclairages précieux sur les choix d'architecture et d'approche à adopter pour la classification d'images, en fonction des caractéristiques spécifiques des données et des exigences du problème. Ces informations pourront servir de base pour des travaux futurs dans le domaine de la vision par ordinateur et contribuer à l'amélioration des performances de classification d'images..

Chapitre 1

Introductions aux Images

1.1 Définitions et caractéristiques d'image numérique.

Une image numérique est une transformation d'une image réelle à travers différents outils traitement (caméras, scanners, satellites, etc.). Cette image numérique est composée de pixels chacun contient des informations différentes (intensité lumineuse, couleur, etc...).

Autrement dit, une image peut être décrite comme une fonction bidimensionnelle $f(x,y)$, où les coordonnées spatiales x et y sont associées à l'intensité ou au niveau de gris de chaque point (x,y) . Lorsque cette fonction est discrétisée en une grille de points, on obtient un nombre ou image de nombres, parfois noté par la lettre I , où les coordonnées (x,y) sont remplacées par (i,j) . Un pixel ou un élément d'image est à peu près un point rectangulaire ou carré qui représente le plus petit bloc de construction d'une image numérique. Les dimensions des pixels peuvent être modifiées en ajustant l'écran ou la carte graphique, et des ensembles de pixels sont stockés dans un tableau à deux dimensions représentant l'image.[1]

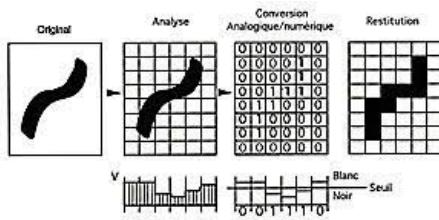


FIGURE 1.1 – Représentation d'une image numérique

1.2 Types des images numériques

Dans le domaine de l'imagerie numérique, nous pouvons distinguer trois types d'images :

1.2.1 Images binaires

une image binaire est une image qui ne peut prendre que deux valeurs possibles pour chaque pixel. Donc elle s'agit d'une image noire et blanche. Chaque pixel est soit noir, soit blanc. Il faut alors un seul bit pour coder un pixel (0 pour noir, 1 pour blanc).

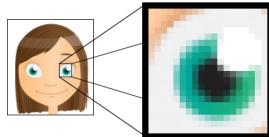


FIGURE 1.2 – Représentation d'une image binaire

1.2.2 Image en niveaux de gris

Dans une image en niveaux de gris, les pixels de l'image sont Exprimé sur 8 bits (1 octet). Alors nous avons 256 possibilités (Disons 256 niveaux de gris). Une valeur de 0 correspond au noir et Une valeur de 255 correspond au blanc. La valeur médiane est Gris plus ou moins foncé.

1.2.3 Image couleur (ou RGB)

Chaque pixel reçoit directement les valeurs de trois canaux RVB (rouge-vert-bleu, RVB), chaque composant RVB nécessite un octet (8bits) capable d'afficher 256 canaux d'intensités différentes, Ainsi, chaque pixel sera représenté par 24 bits.

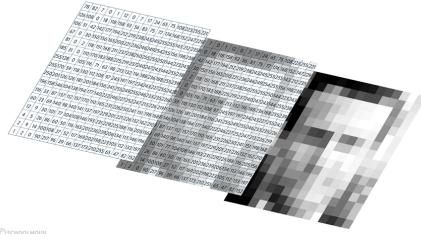


FIGURE 1.3 – Représentation image en niveaux de gris

cela nous permet d'obtenir finalement $256 * 256 * 256 = 16\ 777\ 216$ couleurs différentes.

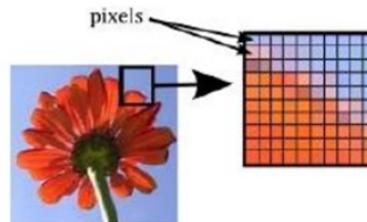


FIGURE 1.4 – Représentation image couleur

1.3 Formats de fichier et compression d'image

Les formats les plus utilisés sont :

1.3.1 BMP (bitmap)

Un bitmap est un tableau qui contient les couleurs de chaque pixel d'une image. Un fichier BMP contient une image non compressée et un entête de 54 octets qui contient les paramétrages de l'image. Ensuite, les composantes RGB (rouge-vert-bleu) de chaque pixel sont stockées. Par exemple, un fichier BMP pour une image de 800×600 pixels a une taille de 1 440 054 octets. Pour économiser de l'espace, la plupart des images sont compressées dans des formats tels que JPEG ou PNG.

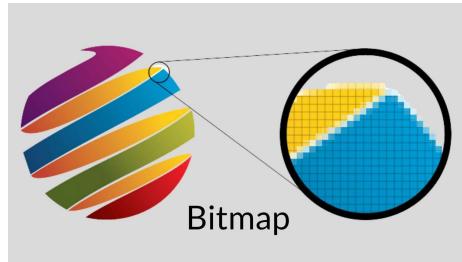


FIGURE 1.5 – Représentation d'une image en bitmap

1.3.2 GIF (GraphicInterchange Format)

GIF est un format de fichier numérique qui utilise une compression de données sans perte, ce qui signifie qu'aucune information n'est perdue pendant le processus de compression.

1.3.3 JPEG (Joint Photo Expert Group)

Le format JPEG permet de compresser les images avec perte. Cette compression peut réduire la taille du fichier initial jusqu'à 20 fois. Cependant, certaines informations sont supprimées lors de la compression et ne peuvent être récupérées lors de la décompression. Cela peut altérer la qualité de l'image.

En plus des formats JPEG, il existe d'autres formats couramment utilisés pour les images tels que FPX (Flashpix), PCD (Photo CD), PNG (Portable Network Graphic), PSD (PhotoShop Document), PSP(Paint Shop Pro) et TIF (Tagged Image File Format).

1.4 Espaces de couleurs

1.4.1 Modèles de couleur (RVB, CMJN, etc.)

• RVB (Rouge, Vert, Bleu)

Le terme RVB (en anglais pour "red, green, blue") décrit l'espace colorimétrique vu par l'œil humain, créé par synthèse additive des trois couleurs primaires. Ce modèle est utilisé par défaut sur les écrans d'ordinateur. Dans ce modèle, chaque couleur a une valeur d'intensité comprise entre 0 et 255. En faisant varier ces valeurs, de nombreuses teintes peuvent être créées.

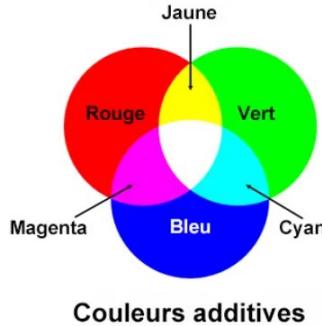


FIGURE 1.6 – modèles de couleur RVB

• CMJN

Le système de couleurs CMJN, également appelé "cyan, magenta, yellow, key", est l'un des espaces colorimétriques les plus couramment utilisés pour représenter les couleurs dans les documents imprimés. Il est basé sur la composition soustractive des couleurs, qui mélange les couleurs en soustrayant la lumière, par opposition à la composition additive, qui ajoute de la lumière. Dans un système CMJN, les trois primaires sont le cyan, le magenta et le jaune, qui peuvent être mélangés pour produire une grande variété de couleurs. Cependant, il est souvent nécessaire d'ajouter du noir pour créer des nuances plus profondes et plus sombres d'une couleur donnée. L'espace colorimétrique CMJN est utilisé dans de nombreux secteurs, notamment l'impression commerciale, la photographie, l'édition de magazines et de journaux et l'emballage. Sa large gamme de couleurs et sa capacité à reproduire avec précision les tons et les textures de la peau en font un choix populaire pour de nombreuses applications.

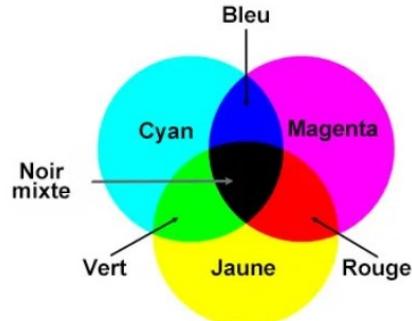
1.5 Applications aux traitements d'image

1.5.1 Système de traitement d'images

Un système de traitement numérique d'images est composé de :

1.5.2 Pré-traitement d'images

Le pré-traitement d'images est une étape qui vise à améliorer la qualité des images numériques et à en extraire de l'information utile pour des applications spécifiques. Il s'agit d'un ensemble de techniques qui permettent de corriger les défauts ou les distorsions introduits par l'acquisition, la numérisation, la transmission ou le stockage des images. Le pré-traitement d'images peut comprendre des opérations telles que : [2]



Couleurs soustractives

FIGURE 1.7 – modèles de couleur CMJN

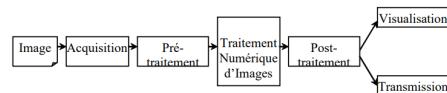


FIGURE 1.8 – Schéma d'un système de traitement d'images

La restauration

elle consiste à réduire le bruit, le flou, les artefacts ou les dégradations présents dans les images, en utilisant des modèles statistiques ou physiques du processus de dégradation1.

L'égalisation

elle vise à améliorer le contraste ou la luminosité des images, en ajustant la distribution des niveaux de gris ou des couleurs1.

La segmentation

elle consiste à diviser les images en régions homogènes ou en objets d'intérêt, en utilisant des critères de similarité ou de discontinuité1.

La détection de contours

elle permet d'extraire les bordures des objets ou des régions dans les images, en utilisant des opérateurs de dérivée ou de gradient1.

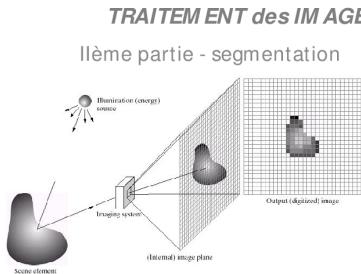


FIGURE 1.9 – Pré-traitement d’images

La filtration

elle sert à éliminer ou à renforcer certaines fréquences spatiales dans les images, en utilisant des filtres linéaires ou non linéaires¹.

1.6 Conclusion.

Le chapitre d’introduction sur les images numériques fournit une vue d’ensemble des concepts clés liés aux images numériques. Il aborde les définitions et caractéristiques des images numériques, les différents types d’images, les espaces de couleurs utilisés pour représenter les images et les applications des traitements d’images.

Dans ce chapitre, nous avons appris que les images numériques sont des représentations visuelles stockées sous forme de données binaires, composées de pixels. Les différentes caractéristiques d’une image, telles que la résolution et la profondeur de couleur, ont été expliquées. Nous avons également été introduits aux différents types d’images numériques, qui peuvent être des images en niveaux de gris, des images en couleur ou des images binaires. Les espaces de couleurs ont été mentionnés comme des systèmes permettant de représenter les couleurs dans les images, tels que RGB et CMYK.

Enfin, le paragraphe a souligné les applications des traitements d’images, qui incluent la correction d’images, la segmentation d’objets, la reconnaissance de formes et bien d’autres. Les traitements d’images jouent un rôle essentiel dans de nombreux domaines tels que la vision par ordinateur, la médecine, la surveillance, l’industrie du divertissement, etc.

Chapitre 2

Intelligence artificielle, Machine learning et Deep learning

2.1 Introduction

Avant de commencer, il est essentiel de clarifier les termes que nous utilisons lorsqu'il s'agit de parler d'IA. Nous devons comprendre la signification de l'intelligence artificielle, de l'apprentissage automatique et de l'apprentissage en profondeur. Pour mieux visualiser ces concepts, vous pouvez vous référer à la figure ci-dessous :

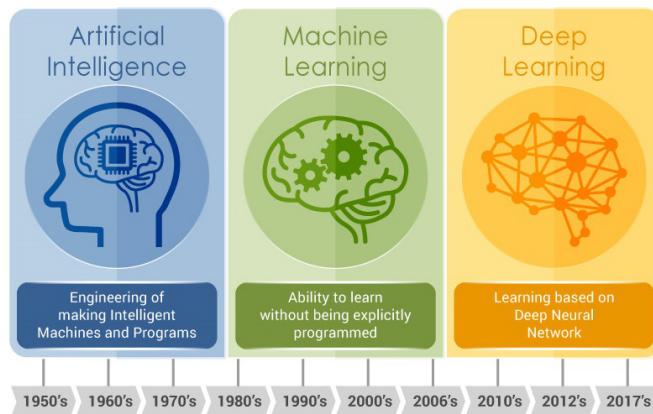


FIGURE 2.1 – Intelligence artificielle, Machine learning et Deep learning

2.2 Intelligence artificielle

L'intelligence artificielle (IA) est un domaine qui fusionne l'informatique avec des ensembles de données robustes afin de résoudre des problèmes. Son objectif est de développer des machines ou des programmes capables de reproduire ou d'imiter certaines fonctions cognitives humaines. L'IA englobe divers types et niveaux en fonction des objectifs et des méthodes utilisés.[3]

2.3 Machine learning

La Machine Learning (ou apprentissage automatique en français) est une branche de l'intelligence artificielle (IA) axée sur la création de systèmes qui apprennent ou fonctionnent mieux en fonction des données qu'ils traitent. L'intelligence artificielle est un terme général qui fait référence à tout système ou machine qui simule une forme d'intelligence humaine. L'apprentissage automatique et l'IA sont souvent abordés ensemble et ne font pas exactement référence au même concept, mais les termes sont parfois utilisés de manière interchangeable. Une différence importante est que l'apprentissage automatique dépend entièrement de l'intelligence artificielle, mais l'intelligence artificielle ne se limite pas à l'apprentissage automatique. Dans l'apprentissage automatique, les algorithmes sont formés pour trouver des modèles et des corrélations dans de grands ensembles de données et prendre les meilleures décisions et prédictions sur la base de cette analyse. La pratique améliore les applications d'apprentissage automatique. Et plus il y aura de données.

2.3.1 Les types de machine learning

La théorie de l'apprentissage utilise des outils mathématiques dérivés de la théorie des probabilités et de la théorie de l'information. Cela vous permet d'évaluer l'optimalité de certaines méthodes par rapport aux autres.

Les algorithmes jouent un rôle important dans l'apprentissage automatique. On peut citer trois types d'algorithme d'apprentissage automatique :

- **Apprentissage supervisé**

Les algorithmes d'apprentissage automatique supervisé sont les plus couramment utilisés. Dans ce modèle, l'humain agit comme un guide, indiquant à l'algorithme les conclusions qu'il doit tirer. Tout comme un enfant apprend à reconnaître des fruits en les mémorisant dans un livre d'images, dans l'apprentissage supervisé, les algorithmes apprennent grâce à des ensembles de données déjà étiquetés et leurs résultats sont prédéfinis.

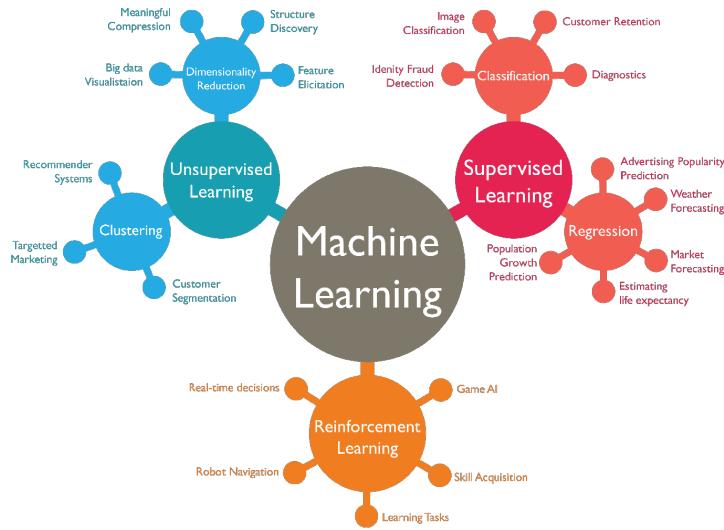


FIGURE 2.2 – Types de Machine Learning

Des exemples d'apprentissage automatique supervisé incluent des algorithmes tels que la régression linéaire et logistique, la classification multicatégorie et les machines à vecteurs de support.

● Apprentissage non supervisé

L'apprentissage automatique non supervisé utilise une approche plus indépendante. Avec cette approche, les ordinateurs apprennent à reconnaître des processus et des modèles complexes sans avoir constamment besoin de conseils humains rigoureux. L'apprentissage automatique non supervisé implique une formation sur des données sans étiquettes ni résultats spécifiquement définis.

Poursuivant l'analogie de la scolarité, l'apprentissage automatique non supervisé s'apparente à aider un enfant à reconnaître les fruits en observant leurs couleurs et leurs motifs, plutôt qu'à mémoriser leurs noms avec l'aide d'un enseignant. L'enfant cherche des similitudes dans les images, les divise en groupes et donne à chaque groupe sa propre étiquette. Des exemples d'algorithmes d'apprentissage automatique non supervisés sont le clustering k-means, l'analyse des composants principaux et indépendants et les règles d'association.

● Apprentissage par renforcement

Les algorithmes doivent apprendre en interagissant avec l'environnement. L'algorithme reçoit une récompense ou une pénalité en fonction de ses actions

et doit apprendre à ajuster ses actions pour maximiser la récompense.[4]

2.4 Deep learning

L'apprentissage en profondeur (deep learning) est un domaine de recherche sur l'apprentissage automatique basé sur un type particulier de mécanisme d'apprentissage.

Il est caractérisé par l'effort de créer un modèle d'apprentissage à plusieurs niveaux, dans lequel les niveaux les plus profonds prennent en compte les résultats des niveaux précédents, les transformant et en faisant toujours plus d'abstraction. Cet aperçu des niveaux d'apprentissage est inspiré par la façon dont le cerveau traite l'information et apprend en réagissant aux stimuli externes. Chaque niveau d'apprentissage correspond, par hypothèse, à l'une des différentes zones qui composent le cortex cérébral.

2.5 La relation entre la Machine Learning et Deep Learning

L'apprentissage profond est en fait une technique d'apprentissage automatique, plus précisément une technique d'apprentissage supervisé basée sur des réseaux de neurones artificiels multicouches. Un réseau de neurones est un modèle mathématique qui imite le fonctionnement des neurones dans le cerveau humain. Un réseau neuronal profond est un réseau neuronal qui contient plusieurs couches cachées, des couches de neurones qui ne sont pas directement connectées aux entrées ou sorties du réseau. Les réseaux de neurones profonds peuvent représenter des fonctions très complexes et capturer des caractéristiques abstraites des données d'entrée. L'apprentissage en profondeur a révolutionné de nombreux domaines tels que la vision par ordinateur, la reconnaissance vocale et le traitement du langage naturel.

En fait, ces domaines nécessitent souvent le traitement de données non structurées et complexes telles que des images, de la parole et du texte. Cependant, l'apprentissage en profondeur n'est qu'une des nombreuses techniques d'apprentissage automatique disponibles. D'autres techniques telles que les arbres de décision, les méthodes de régression, les méthodes bayésiennes et même les SVM (Support Vector Machines) sont également couramment utilisées dans diverses situations. [5]

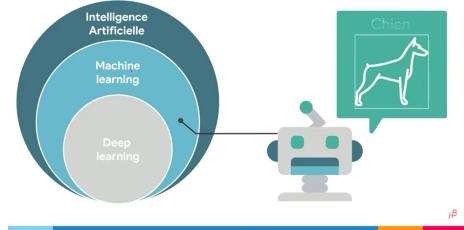


FIGURE 2.3 – Machine Learning et Deep Learning

2.6 Brève description d’algorithme populaire

2.6.1 Neural Networks

Les réseaux de neurones sont un autre algorithme d’apprentissage automatique et ils ont connu des périodes de grande popularité et des périodes pendant lesquelles ils étaient rarement utilisés.

Le cerveau est composé de nombreux neurones qui se transmettent de l’information sous forme d’impulsions électriques. Pour vous donner une idée du fonctionnement des neurones, il y a des filaments appelés dendrites qui reçoivent l’impulsion. Ensuite, le neurone retransmet cette impulsion à travers de longs câbles appelés axones. L’impulsion passe alors par des synapses et continue jusqu’à un autre neurone. Bien évidemment, le cerveau est extrêmement complexe, donc les ingénieurs n’ont pas cherché à programmer un cerveau tel qu’il existe vraiment dans la nature. Ils ont donc créé une abstraction de son fonctionnement, appelée réseau de neurones. Pour expliquer le fonctionnement d’un réseau de neurones, il faut comprendre comment fonctionnent les neurones individuels. [6]

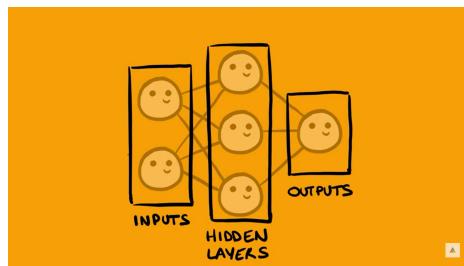


FIGURE 2.4 – Neural networks

• Les fonctions d'activation

Les fonctions d'activation sont utilisées dans les réseaux de neurones pour introduire de la non-linéarité dans leurs calculs. Elles permettent aux neurones de modéliser des relations complexes entre les entrées et les sorties, en introduisant des seuils et des non-linéarités dans la propagation des informations à travers le réseau. Ainsi, les fonctions d'activation sont essentielles pour permettre aux réseaux neuronaux d'apprendre et de représenter des modèles complexes et non linéaires.

Il existe plein de fonctions d'activation mais les plus connu c'est :

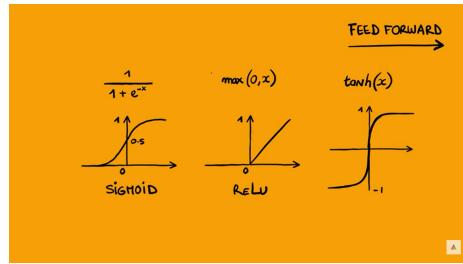


FIGURE 2.5 – Fonctions d'activation

2.7 Conclusion

Nous avons consacré ce chapitre à la présentation des notions de base comme L'IA , l'apprentissage automatique et ses types et on conclut avec l'algorithme la plus populaire et la plus utilisée.

Chapitre 3

La vision par ordinateur

3.1 Introduction à la vision par ordinateur

La vision par ordinateur vise à permettre aux machines de comprendre les images de manière similaire à la perception humaine. La classification des images est une tâche essentielle dans ce domaine, permettant d'attribuer des catégories aux images. Elle utilise des techniques avancées de traitement d'images et d'apprentissage automatique pour résoudre des problèmes complexes dans divers secteurs.

3.2 Les fondements de la vision par ordinateur

3.2.1 Système visuel humain et perception visuelle

La vision par ordinateur est une intelligence artificielle qui permet aux ordinateurs et aux systèmes de dériver des informations significatives à partir d'images numériques, de vidéos et d'autres entrées visuelles et de prendre des mesures ou de faire des recommandations sur la base de ces informations. Si l'intelligence artificielle permet aux ordinateurs de penser, la vision par ordinateur leur permet de voir, d'observer et de comprendre.

La vision par ordinateur fonctionne de la même manière que le système visuel humain (est l'ensemble des organes participant à la perception visuelle humaine). les systèmes de la vision par ordinateur ont été développés pour reconnaître et extraire des informations visuelles à partir d'images et de vidéos à l'aide d'algorithmes qui imitent les processus de traitement visuel du cerveau humain.

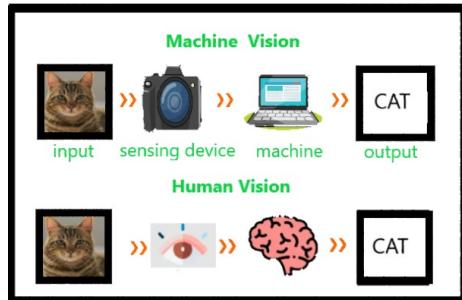


FIGURE 3.1 – Machine Vision VS Human Vision

3.2.2 Fondements théoriques de la vision par ordinateur

Les fondements théoriques de la vision par ordinateur qu'on utilise sont basés sur des modèles mathématiques et des algorithmes d'intelligence artificielle qui permettent aux machines de traiter et d'analyser des données visuelles, comme des images ou des vidéos. Par exemple, on utilise des réseaux de neurones convolutifs (CNN) pour apprendre à la machine à reconnaître des objets, des visages, des scènes, etc., à partir de millions d'images étiquetées¹. On utilise aussi des techniques de géométrie projective pour reconstruire la profondeur et la structure 3D d'une scène à partir de plusieurs vues². La vision par ordinateur a de nombreuses applications dans différents domaines, comme la médecine, la sécurité, l'automobile, etc...

3.3 Classification d'images : Concepts et méthodes

3.3.1 Introduction à la classification des images

Le concept de la classification des images est de répartir systématiquement des images selon des classes établies au préalable¹. Par exemple, on peut classer des images de fruits selon leur couleur, leur forme ou leur espèce. La classification des images est une étape fondamentale de la vision par ordinateur, car c'est elle qui permet de décrire ce que contient une image. Il existe deux types principaux de classification des images : la classification non supervisée et la classification supervisée. La classification non supervisée consiste à utiliser un algorithme qui apprend par lui-même à analyser et à regrouper les données brutes. La classification supervisée consiste à utiliser un algorithme qui est entraîné avec des données étiquetées au préalable.[7]

3.3.2 Méthodes de classification des images

Il existe plusieurs méthodes de classification des images, mais on peut les regrouper en deux grandes catégories : la classification non supervisée et la classification supervisée¹. La classification non supervisée consiste à utiliser un algorithme qui apprend par lui-même à analyser et à regrouper les données brutes. La classification supervisée consiste à utiliser un algorithme qui est entraîné avec des données étiquetées au préalable. Ces deux méthodes peuvent être basées sur des objets ou sur des pixels. La classification basée sur les objets utilise des segments d'images comme unités de base, tandis que la classification basée sur les pixels utilise des pixels individuels comme unités de base. La classification des images peut être un workflow long avec plusieurs étapes de traitement.

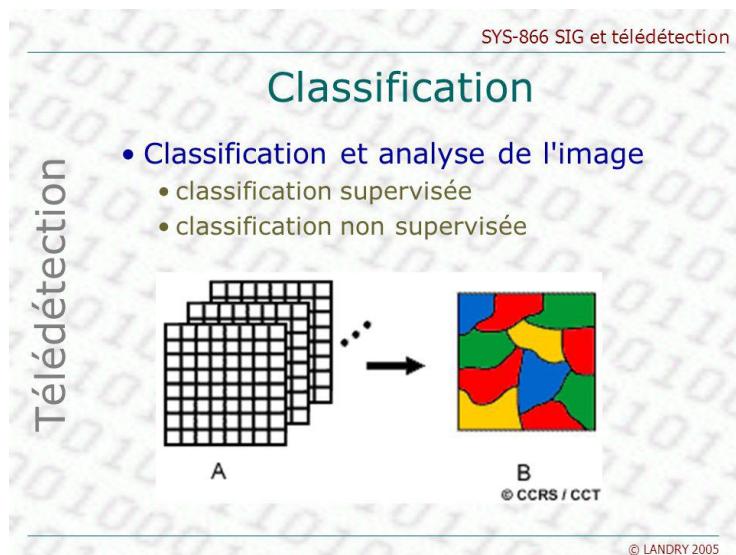


FIGURE 3.2 – Classification des images

3.3.3 Entraînement des modèles de classification d'images

L'entraînement des modèles de classification d'images consiste à apprendre à un algorithme à assigner correctement une catégorie à une image en utilisant des données d'entraînement étiquetées. Les réseaux neuronaux sont utilisés pour reconnaître des motifs et extraire des caractéristiques pertinentes pour la classification. Des bibliothèques comme PyTorch et Keras facilitent l'entraînement des modèles en permettant la création, la configuration, l'entraînement, l'évaluation et le déploiement des réseaux neuronaux. Des modèles pré-entraînés, tels que InceptionV, sont également disponibles pour des tâches

spécifiques, ayant été entraînés sur de grandes bases de données d'images

3.4 Conclusion

En résumé, ce chapitre a exploré les fondements de la vision par ordinateur en se concentrant sur les modèles mathématiques et les algorithmes d'intelligence artificielle utilisés pour traiter et analyser les données visuelles. Nous avons également examiné l'importance du système visuel humain et de la perception visuelle dans la compréhension des images. Enfin, nous avons abordé les concepts et les méthodes de la classification d'images, mettant en évidence son rôle clé dans la reconnaissance d'objets, de visages et de scènes, ainsi que ses applications dans divers domaines. Malgré les progrès réalisés, des défis subsistent et de nouvelles avancées sont attendues dans le domaine de la vision par ordinateur.

Chapitre 4

Classification des images à l'aide de CNN et GNN

4.1 Réseaux des neurones convolutifs (CNN)

Un réseau neuronal convolutif (ConvNet ou CNN) est un réseau neuronal artificiel (ANN) qui utilise des algorithmes d'apprentissage en profondeur pour analyser des images, classer des éléments visuels et effectuer des tâches de vision par ordinateur.

Les CNN utilisent des principes d'algèbre linéaire tels que la multiplication matricielle pour reconnaître les modèles dans les images. Ces processus impliquent des calculs complexes, de sorte qu'une unité de traitement graphique (GPU) est nécessaire pour entraîner le modèle.

En termes simples, les CNN utilisent des algorithmes d'apprentissage en profondeur pour prendre des données d'entrée, telles que des images, et attribuer une signification à différents aspects de cette image sous la forme de biais et de poids apprenables. Cela permet au CNN de distinguer ou de classer les images.[8]

Il y a quatre types de couches pour un CNN :

- **La couche de convolution**

La couche de convolution joue un rôle crucial dans les réseaux de neurones convolutifs en détectant les caractéristiques spécifiques dans les images. Elle applique une opération de convolution où des filtres glissent sur l'image pour calculer les produits de convolution. Chaque filtre est conçu pour détecter des caractéristiques spécifiques. Cette opération génère des cartes d'activation qui indiquent les emplacements où les caractéristiques sont présentes dans l'image. Par exemple, en appliquant la convolution entre une image et une matrice de filtres, on obtient une "Feature Map" qui met en évidence les régions où les caractéristiques sont détectées.

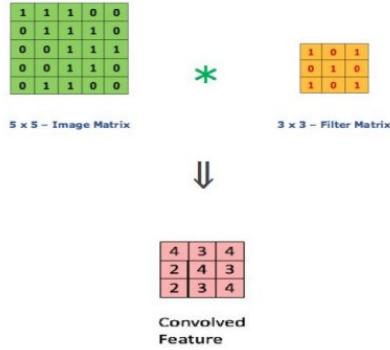


FIGURE 4.1 – La couche de Convolution

• La couche de pooling

La couche de pooling est positionnée entre deux couches de convolution dans un réseau de neurones convolutifs. Elle réduit la taille des cartes de caractéristiques tout en préservant les informations importantes. Pour cela, elle découpe l'image en cellules régulières et conserve la valeur maximale de chaque cellule. En utilisant des cellules carrées de petite taille, on minimise la perte d'informations. En sortie, on obtient des cartes de caractéristiques de taille réduite, ce qui réduit les paramètres et les calculs du réseau, améliorant ainsi son efficacité. L'objectif principal de la couche de pooling est de réduire la dimensionnalité des données, ce qui favorise une meilleure généralisation du modèle, une réduction du surapprentissage et une amélioration de l'efficacité en termes de calculs et de mémoire.

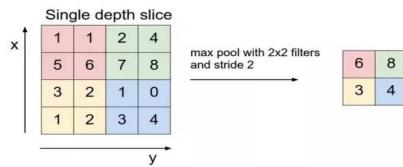


FIGURE 4.2 – La Couche de Pooling

• La couche ReLU

C'est une fonction non linéaire qui est définie comme suit : $\text{ReLU}(x) = \max(0, x)$.

Dans une couche de ReLU, toutes les valeurs négatives de l'entrée sont remplacées par zéro, tandis que les valeurs positives restent inchangées. Cette opération est réalisée point par point sur chaque élément de l'entrée.

La couche ReLU joue un rôle crucial en tant que fonction d'activation dans les réseaux de neurones. Elle permet d'introduire de la non-linéarité dans le modèle, ce qui est essentiel pour qu'un réseau puisse apprendre des relations complexes entre les caractéristiques. En supprimant les valeurs négatives, la couche ReLU aide à introduire de la sparsité dans l'activation des neurones, ce qui peut conduire à une meilleure représentation des caractéristiques discriminantes.

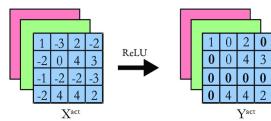


FIGURE 4.3 – La couche ReLU

- **La couche fully-connected**

La couche fully-connected est la dernière couche présente dans les réseaux de neurones. Chaque neurone est connecté à tous les neurones de la couche précédente. Les entrées de la couche précédente sont pondérées, sommées et passées à travers une fonction d'activation pour obtenir les activations des neurones de la couche fully-connected. Cette couche permet de capturer les relations complexes entre les caractéristiques extraites des couches précédentes. Elle est utilisée pour des tâches de classification, de régression et d'autres types de prédictions en utilisant les caractéristiques extraites comme entrée.

4.2 Les réseaux de neurones graphiques (GNN)

Les Graph Neural Networks (GNN) sont une classe de modèles d'apprentissage automatique conçus pour traiter des données structurées sous forme de graphes. Ils ont été initialement développés pour des tâches de classification et de prédiction sur des données de graphes, mais ils ont également été étendus avec succès à d'autres types de données, y compris les images.

La classification des images à l'aide de GNN consiste à représenter une image sous forme de graphe, où les noeuds représentent des régions ou des pixels de l'image, et les arêtes capturent les relations spatiales ou contextuelles entre ces régions. Les GNN sont ensuite utilisés pour apprendre des représentations des noeuds et des arêtes du graphe, qui sont ensuite utilisées pour prédire les étiquettes de classification des images. Dans les Graph Neural Networks (GNN)

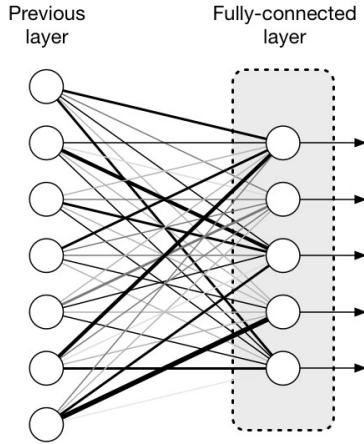


FIGURE 4.4 – La Couche fully-connected

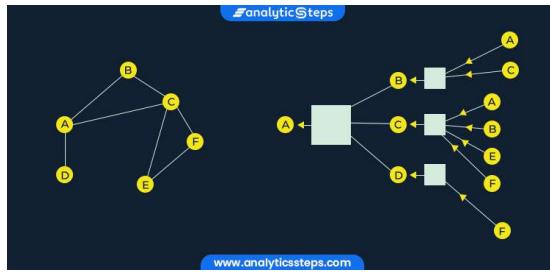


FIGURE 4.5 – Les réseaux de neurones graphiques

pour la classification d’images, on utilise généralement deux types principaux de couches :

- **Couche de propagation des graphes (GCN)**

Cette couche est responsable de la propagation des informations à travers les noeuds et les arêtes du graphe. Elle calcule les nouvelles représentations des noeuds en agrégeant les informations des noeuds voisins. Les opérations de convolution dans les GNN sont adaptées pour travailler sur des graphes au lieu de matrices d’images.

- **Couche entièrement connectée (Fully-Connected Layer)**

Cette couche est utilisée pour combiner les caractéristiques extraites des couches précédentes et effectuer la classification finale. Elle prend les représentations des noeuds obtenues après les couches de propagation des graphes et les mappe vers les classes de sortie.

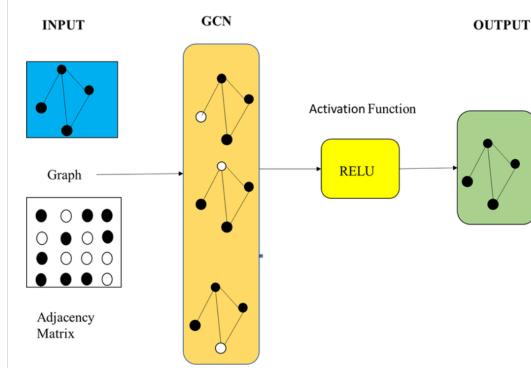


FIGURE 4.6 – Les Couches de convolution graphique

Il existe également d'autres types de couches spécifiques utilisées dans les GNN, telles que les couches de regroupement (pooling) pour réduire la taille du graphe, les couches de régularisation pour éviter le surapprentissage, et les couches d'attention pour mettre l'accent sur les nœuds importants du graphe. La combinaison de ces différentes couches permet aux GNN de capturer des informations contextuelles et de réaliser une classification précise des images représentées sous forme de graphes.

4.3 Conclusion

En conclusion, les réseaux de neurones convolutifs (CNN) et les réseaux de neurones graphiques (GNN) sont deux approches puissantes pour l'analyse et la classification d'images. Les CNN utilisent des couches de convolution, de pooling, de ReLU et fully-connected pour extraire et interpréter les caractéristiques visuelles des images, leur permettant de distinguer et de classifier différents objets. Les GNN, quant à eux, exploitent la structure en graphe des images en utilisant des couches de propagation des graphes et des couches fully-connected pour apprendre les relations entre les nœuds et les arêtes, ce qui leur permet de prédire les étiquettes de classification des images représentées sous forme de graphes. Ces deux approches offrent des solutions efficaces pour la vision par ordinateur et ouvrent des perspectives prometteuses dans de nombreux domaines d'application.

Chapitre 5

Implementation

5.1 Environnement et langages de développement

5.1.1 Python

[10] Python est un langage de programmation haut niveau, multiplateforme, favorisant une programmation structurée, fonctionnelle, orientée objet et impérative. Il propose un typage dynamique puissant, une gestion automatique de la mémoire avec récupération de place et un système de gestion des exceptions. Python est largement utilisé pour le développement de logiciels, les applications Web, la science des données et l'apprentissage automatique.

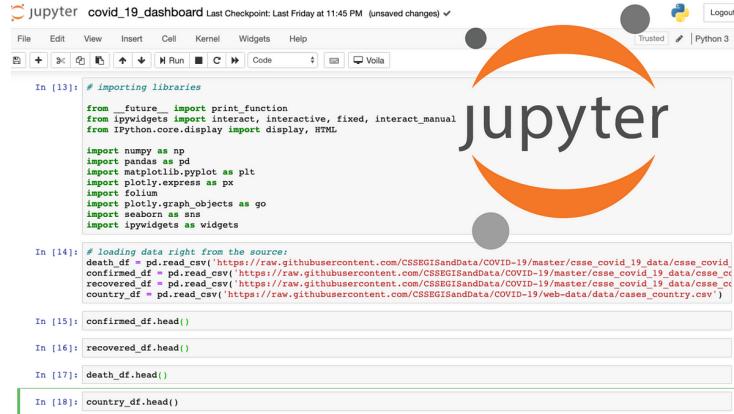


FIGURE 5.1 – Python

5.1.2 Jupyter Notebook

Jupyter Notebook est une application web qui permet de créer des documents interactifs combinant du code exécutable, des visualisations et des explications textuelles. Utilisé pour la classification d'images, Jupyter Notebook vous permettra de développer, d'exécuter et de documenter votre code Python pour la création et l'entraînement de modèles de machine

learning. Vous pourrez visualiser les résultats, présenter les images et ajouter des explications détaillées, le tout dans un environnement interactif et convivial.



The screenshot shows a Jupyter Notebook interface with the title "jupyter covid_19_dashboard". The notebook has several code cells:

- In [13]:

```
# importing libraries
from __future__ import print_function
from ipywidgets import interact, interactive, fixed, interact_manual
from IPython.core.display import display, HTML
```
- In [14]:

```
# loading data right from the source:
death_df = pd.read_csv('https://raw.githubusercontent.com/CSSEGISandData/COVID-19/master/csse_covid_19_data/csse_covid_19_deaths.csv')
confirmed_df = pd.read_csv('https://raw.githubusercontent.com/CSSEGISandData/COVID-19/master/csse_covid_19_data/csse_covid_19_confirmed.csv')
recovered_df = pd.read_csv('https://raw.githubusercontent.com/CSSEGISandData/COVID-19/master/csse_covid_19_data/csse_covid_19_recovered.csv')
country_df = pd.read_csv('https://raw.githubusercontent.com/CSSEGISandData/COVID-19/web-data/data/cases_country.csv')
```
- In [15]:

```
confirmed_df.head()
```
- In [16]:

```
recovered_df.head()
```
- In [17]:

```
death_df.head()
```
- In [18]:

```
country_df.head()
```

FIGURE 5.2 – Jupyter

5.2 Bibliothèques utilisées

5.2.1 PyTorch

PyTorch est un framework (bibliothèque logicielle) open-source de machine learning largement utilisé pour Python. Il est basé sur Torch, une bibliothèque de calcul scientifique qui utilise le langage de programmation Lua. PyTorch a été développé par Facebook et est utilisé pour des applications telles que la vision par ordinateur et le traitement du langage naturel.



FIGURE 5.3 – Pytorch

• Torch

Torch est la bibliothèque principale de PyTorch. Elle est utilisée pour l'informatique tensorielle et la création de réseaux de neurones.

- **Torch.nn**

Torch.nn est une sous-bibliothèque de PyTorch qui fournit des classes et des fonctions pour la construction de réseaux de neurones.

- **Torch.nn.Functional**

Torch.nn.Functional est une sous-bibliothèque de PyTorch contenant des fonctions d'activation et d'autres opérations mathématiques utilisées dans les réseaux de neurones.

- **Torchvision**

Torchvision est une extension spécifique à la vision par ordinateur de PyTorch qui fournit des utilitaires de chargement de jeux de données d'images et de transformation d'images.

- **Torchvision.transforms**

Torchvision.transforms est une sous-bibliothèque de Torchvision qui fournit des classes pour effectuer des transformations sur des images.

5.2.2 Matplotlib

Matplotlib est une bibliothèque de Python utilisée pour visualiser les données. Elle permet de créer des graphiques en 2D et en 3D afin de mieux comprendre les données. Matplotlib est particulièrement utile pour la visualisation de données scientifiques et statistiques. Grâce à ses nombreuses fonctionnalités, elle offre la possibilité de créer des graphiques de lignes, des diagrammes à barres, des histogrammes, des diagrammes de dispersion, des graphiques en secteurs, des graphiques en boîte et bien plus encore. Avec Matplotlib, les chercheurs peuvent représenter visuellement leurs résultats, explorer des tendances, détecter des modèles et communiquer efficacement leurs découvertes.

5.3 Implémentation du modèle CNN

5.3.1 Le jeu de donnée utilisé

Le jeu de données MNIST est un ensemble de données largement utilisé en apprentissage automatique et en vision par ordinateur. Il contient des images

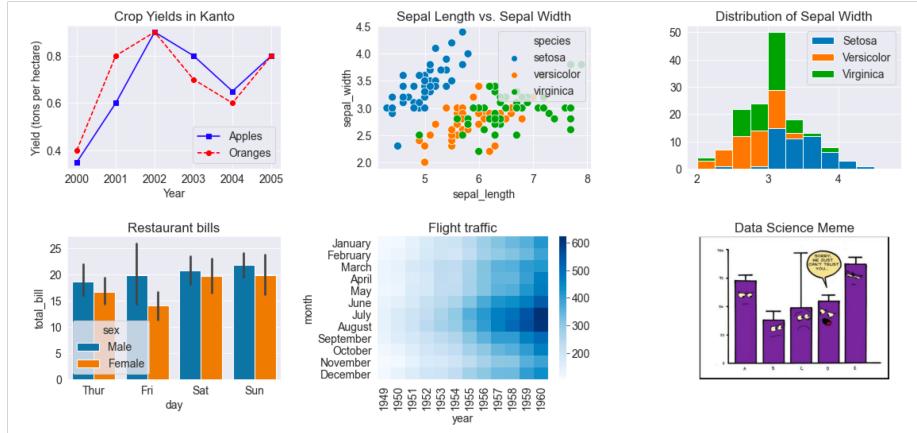


FIGURE 5.4 – matplotlib

en niveaux de gris de chiffres écrits à la main, allant de 0 à 9.

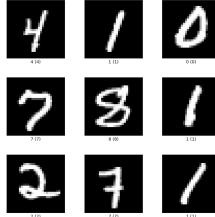


FIGURE 5.5 – MNIST dataset

Dans notre code ,On a utilisé la bibliothèque torchvision pour télécharger et charger le jeu de données MNIST. Ce jeu de données est séparé en deux parties : l'ensemble d'entraînement et l'ensemble de test. On a effectué des transformations qui permettent de prétraiter les images du jeu de données MNIST en les convertissant en tenseurs et en les normalisant.

Pour l'ensemble d'entraînement, on a spécifié qu'il s'agit de l'ensemble d'entraînement (train=True) et on a appliqué une transformation pour convertir les images en tenseurs. Ensuite, on a créé un DataLoader (train-loader) qui divise les données en petits groupes appelés lots. Chaque lot contient 32 images (BATCH-SIZE). Les images sont mélangées aléatoirement à chaque époque d'entraînement pour aider le modèle à généraliser de manière efficace. De plus, pour charger les données de manière plus rapide, on a utilisé 2 travailleurs. De même, pour l'ensemble de test, on a spécifié qu'il s'agit de l'ensemble de test (train=False). On a également créé un DataLoa-

```

BATCH_SIZE = 32

# Définir les transformations pour les images
transform = transforms.Compose([transforms.ToTensor()])

# Télécharger l'ensemble de données MNIST et le charger
trainset = torchvision.datasets.MNIST(root='C:/Users/marya/OneDrive/Bureau/pfe/mnist/MNIST', train=True,
                                      download=True, transform=transform)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=BATCH_SIZE,
                                          shuffle=True, num_workers=2)
testset = torchvision.datasets.MNIST(root='C:/Users/marya/OneDrive/Bureau/pfe/mnist/MNIST', train=False,
                                      download=True, transform=transform)
testloader = torch.utils.data.DataLoader(testset, batch_size=BATCH_SIZE,
                                         shuffle=False, num_workers=2)

```

FIGURE 5.6 – Téléchargement et prétraitement des images

der (testloader) avec les mêmes paramètres que pour l'ensemble d'entraînement.

En résumé, ce code nous permet de télécharger le jeu de données MNIST, de le diviser en ensembles d'entraînement et de test, et de créer des objets DataLoader pour charger les données en petits groupes afin de les utiliser lors de l'entraînement et de l'évaluation d'un modèle d'apprentissage automatique.

5.3.2 L'architecture du modèle CNN proposé

L'architecture de ce modèle CNN est composée de trois principales parties : la couche de convolution, les couches entièrement connectées (fully connected layers) et la couche de sortie.

Ce code définit une architecture de modèle appelée MyModel. Le modèle comporte trois couches principales : une couche de convolution, suivie de deux couches fully-connected.

La première couche de convolution prend une image en niveaux de gris de taille 28x28x1 en entrée et produit une sortie de taille 26x26x32. Cela signifie que la couche convective utilise 32 filtres de taille 3x3 pour extraire des caractéristiques de l'image d'entrée.

Ensuite, les sorties de la couche de convolution sont passées à travers une fonction d'activation ReLU, qui remplace les valeurs négatives par zéro et garde les valeurs positives inchangées.

Ensuite, le tenseur résultant est aplati en un vecteur de taille 32x26x26 (ou 32x(26x26)) pour être utilisé comme entrée dans la première couche fully-connected.

La première couche fully-connected prend ce vecteur en entrée et le transforme en un vecteur de taille 128. Une fonction d'activation ReLU est également appliquée à cette sortie.

```

class MyModel(nn.Module):
    def __init__(self):
        super(MyModel, self).__init__()

        # Couche de convolution : prend une image de taille 28x28x1 et produit une sortie de taille 26x26x32
        self.conv1 = nn.Conv2d(in_channels=1, out_channels=32, kernel_size=3)

        # Couche fully-connected : prend en entrée une image de taille 26x26x32 aplatie (32x26x26) et produit une sortie de taille 128
        self.d1 = nn.Linear(26 * 26 * 32, 128)

        # Couche fully-connected : prend en entrée un vecteur de taille 128 et produit une sortie de taille 10
        self.d2 = nn.Linear(128, 10)

    def forward(self, x):
        # Passage de l'image à travers la couche de convolution
        x = self.conv1(x)
        x = F.relu(x) # Application de la fonction d'activation ReLU

        # Aplatissement du tenseur en un vecteur
        x = x.flatten(start_dim=1)

        # Passage du vecteur à travers la première couche fully-connected
        x = self.d1(x)
        x = F.relu(x) # Application de la fonction d'activation ReLU

        # Passage du vecteur à travers la deuxième couche fully-connected
        logits = self.d2(x)

        # Application de la fonction softmax pour obtenir des probabilités normalisées
        out = F.softmax(logits, dim=1)

    return out

```

FIGURE 5.7 – La modèle CNN proposé

Enfin, la deuxième couche fully-connected prend le vecteur de taille 128 en entrée et le transforme en un vecteur de taille 10, qui représente les scores ou les logits associés à chaque classe de sortie. Une fonction softmax est ensuite appliquée aux logits pour obtenir des probabilités normalisées.

C'est ainsi que fonctionne le modèle MyModel. Il prend une image en entrée, la passe à travers les différentes couches, applique des fonctions d'activation appropriées et produit une sortie finale contenant les probabilités associées à chaque classe.

5.3.3 Méthodes d'entraînement

Dans ce modèle, on a utilisé le jeu de données d'entraînement pour entraîner notre modèle. On a défini un taux d'apprentissage (learning-rate) et un nombre d'époques (num-epochs). On a initialisé notre modèle, déplacé les calculs sur le GPU si disponible, défini une fonction de perte (criterion) et un optimiseur (optimizer). Ensuite, on a itéré sur chaque époque. À chaque époque, on a effectué une boucle sur les données d'entraînement (trainloader). Pour chaque lot d'images et d'étiquettes, on a transféré les données sur le GPU, effectué une passe avant (forward pass) à travers le modèle pour obtenir les prédictions (logits), calculé la perte en utilisant la fonction de perte, effectué une rétropropagation (backward pass) pour calculer les gradients et mis à jour les paramètres du modèle (optimizer.step()).

On a également suivi la perte d'entraînement (train-running-loss) et la précision d'entraînement (train-acc) à chaque itération pour les utiliser dans l'affichage des résultats. Après chaque époque, on a évalué les performances du modèle sur l'ensemble de test (testloader) en calculant la précision du modèle sur les données de test (test-acc). On a enregistré les précisions d'entraînement et de test à chaque époque pour les afficher ultérieurement.

Enfin, on a utilisé la bibliothèque matplotlib.pyplot pour tracer les courbes de perte d'entraînement et les courbes de précision d'entraînement et de test. Cela nous a permis de visualiser l'évolution de la perte et de la précision du modèle au fil des époques, dans cette méthode d'entraînement, on a utilisé les données d'entraînement pour entraîner le modèle en ajustant les paramètres à chaque itération. On a évalué les performances du modèle sur l'ensemble de test et tracé les courbes de perte et de précision pour évaluer les performances du modèle sur la tâche de classification d'images.

5.3.4 Résultats obtenue

Pour visualiser les résultats on a utilisé la bibliothèque matplotlib :

```
Epoch: 0 | Loss: 1.6129 | Train Accuracy: 85.21%
Test Accuracy: 96.94%
Epoch: 1 | Loss: 1.4946 | Train Accuracy: 96.94%
Test Accuracy: 97.99%
Epoch: 2 | Loss: 1.4840 | Train Accuracy: 97.96%
Test Accuracy: 98.36%
Epoch: 3 | Loss: 1.4790 | Train Accuracy: 98.43%
Test Accuracy: 98.31%
Epoch: 4 | Loss: 1.4757 | Train Accuracy: 98.75%
Test Accuracy: 98.16%
```

FIGURE 5.8 – Test d'accuracy

Les résultats obtenus lors de l'entraînement de notre modèle d'apprentissage automatique sont très encourageants. Au fil des époques, nous avons observé une diminution constante de la perte (loss) et une amélioration significative de la précision (accuracy) à la fois sur l'ensemble d'entraînement et sur l'ensemble de test. Au départ, notre modèle atteignait une précision de 85.21% sur l'ensemble d'entraînement, qui est rapidement passée à 98.75% à la dernière époque. De même, la précision sur l'ensemble de test est passée de 96.94% à 98.16%. Ces résultats démontrent que notre modèle a appris à reconnaître efficacement les

chiffres manuscrits du jeu de données MNIST. De plus, la diminution de la perte au fil des époques indique que notre modèle s'ajuste progressivement aux données d'entraînement. Ces résultats prometteurs confirment l'efficacité de notre modèle dans la classification des images MNIST et nous permettent de conclure que notre approche d'apprentissage automatique est réussie.

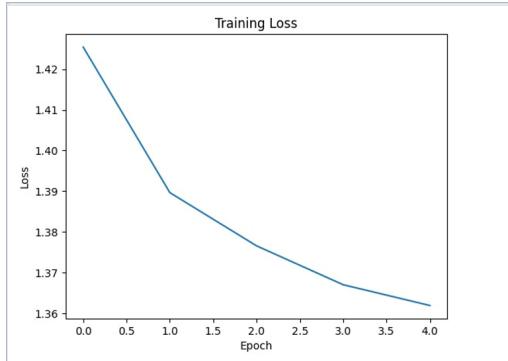


FIGURE 5.9 – Training Loss

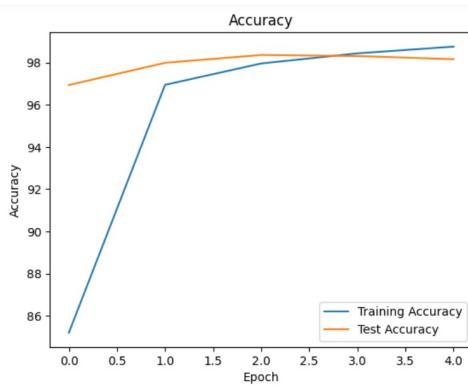
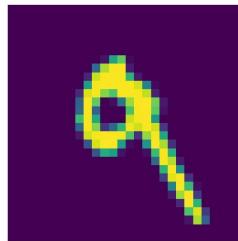


FIGURE 5.10 – Training Accuracy et test Accuracy

Pour tester ces résultats, on a développé une fonction "predict-image".

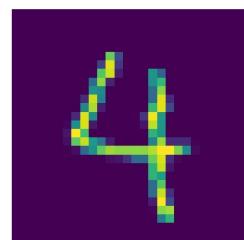
Cette fonction prend une image et notre modèle de réseau de neurones entraîné en entrée. Son objectif est d'effectuer une prédiction sur l'image en utilisant le modèle et de renvoyer l'étiquette prédite correspondante. En utilisant cette fonction, nous avons évalué les performances de notre modèle CNN sur différentes images. Deux exemples de résultats sont présentés dans les figures 5.11 et 5.12. La figure 5.11 montre l'image d'un chiffre manuscrit et le résultat de la prédiction correspondante (nombre 9).



True Label: 9
Predicted Label: 9

FIGURE 5.11 – Résultat (nombre 9)

De même, la figure 5.12 affiche une autre image de chiffre et le résultat prédit (nombre 4). Ces résultats illustrent l'efficacité de notre modèle à classifier avec précision des chiffres manuscrits.



True Label: 4
Predicted Label: 4

FIGURE 5.12 – Résultat (nombre 4)

En conclusion, grâce à la fonction "predict-image" et à notre modèle CNN, nous avons pu obtenir des prédictions précises sur des images individuelles. Ces résultats démontrent l'efficacité des réseaux de neurones convolutifs dans

l'analyse d'images et les tâches de vision par ordinateur.

5.4 Implementation du modèle GNN

5.4.1 Le Jeu de donnée utilisée

Le jeu de données CIFAR-10 se compose de 60000 images en couleurs 32x32 dans 10 classes, avec 6000 images par classe. Il y a 50000 images d'entraînement et 10000 images de test. L'ensemble de données est divisé en cinq lots de formation et un lot de tests, chacun contenant 10 000 images. Le lot de test contient exactement 1000 images sélectionnées au hasard dans chaque classe.

Les lots d'entraînement contiennent les images restantes dans un ordre aléatoire, mais certains lots d'entraînement peuvent contenir plus d'images d'une classe que d'une autre. Entre eux, les lots de formation contiennent exactement 5000 images de chaque classe.

Voici les classes dans le jeu de données, ainsi que 10 images aléatoires de chacune :

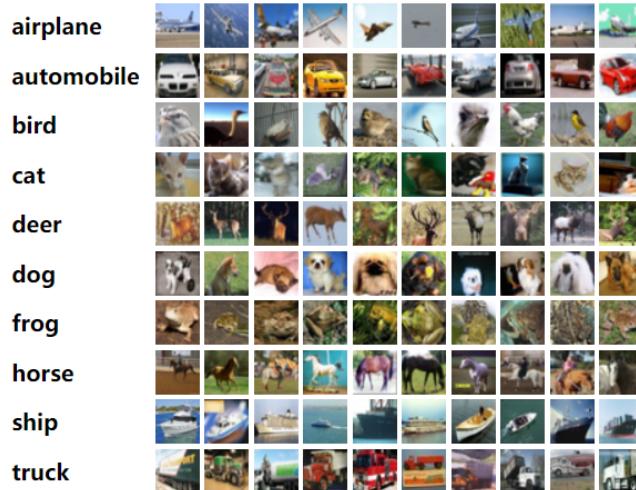


FIGURE 5.13 – CIFAR-10

5.4.2 L'architecture du modèle GNN proposé

L'architecture de ce modèle GNN est composée de trois principales parties : la Couche linéaire , les Couches de convolution graphique (GCN) et la Couche linéaire pour la classification finale.

```

class MyGraphModel(nn.Module):
    def __init__(self, num_nodes):
        super(MyGraphModel, self).__init__()
        # Couche linéaire pour transformer les caractéristiques des noeuds en 3 dimensions
        self.linear = nn.Linear(num_nodes, 3)

        # Couches de convolution graphique (GCN)
        self.conv1 = GCNConv(3, 64) # Première couche de convolution : 3 dimensions en entrée, 64 dimensions en sortie
        self.conv2 = GCNConv(64, 128) # Deuxième couche de convolution : 64 dimensions en entrée, 128 dimensions en sortie
        self.conv3 = GCNConv(128, 3) # Troisième couche de convolution : 128 dimensions en entrée, 3 dimensions en sortie

        # Couche linéaire pour la classification finale
        self.fc1 = nn.Linear(3, 10) # Couche linéaire avec 3 dimensions en entrée et 10 dimensions en sortie

    def forward(self, x, edge_index):
        # Convertir edge_index en une représentation non dirigée du graphe
        edge_index = to_undirected(edge_index)
        # Ajouter des boucles de retour sur soi aux arêtes pour traiter les nœuds isolés
        edge_index, _ = add_self_loops(edge_index, num_nodes=x.size(0))
        # Transformation linéaire initiale des caractéristiques des nœuds
        x = self.linear(x)
        # Appliquer la première couche de convolution graphique (GCN)
        x = self.conv1(x, edge_index)
        x = nn.functional.relu(x) # Appliquer la fonction d'activation ReLU
        # Appliquer la deuxième couche de convolution graphique (GCN)
        x = self.conv2(x, edge_index)
        x = nn.functional.relu(x) # Appliquer la fonction d'activation ReLU
        # Appliquer la troisième couche de convolution graphique (GCN)
        x = self.conv3(x, edge_index)
        x = nn.functional.relu(x) # Appliquer la fonction d'activation ReLU
        # Effectuer un pooling spatial en moyenne sur les dimensions spatiales
        x = nn.functional.avg_pool2d(x, kernel_size=x.size()[2:])
        # Aplatir la sortie pour l'entrée de la couche linéaire
        x = x.view(x.size(0), -1)
        # Appliquer la couche linéaire pour la classification finale
        logits = self.fc1(x)
        out = nn.functional.softmax(logits, dim=1) # Appliquer la fonction de softmax pour obtenir des probabilités de classe
        return out

```

FIGURE 5.14 – L’architecture du modèle GNN

- Couche linéaire : Cette couche est utilisée pour transformer les caractéristiques des nœuds en 3 dimensions. Elle est définie par `self.linear = nn.Linear(num_nodes, 3)`.
- Couches de convolution graphique (GCN) : Le modèle comprend trois couches de convolution graphique. La première couche `self.conv1 = GCNConv(3, 64)` prend en entrée des caractéristiques de 3 dimensions et produit une sortie de 64 dimensions. La deuxième couche `self.conv2 = GCNConv(64, 128)` prend en entrée des caractéristiques de 64 dimensions et produit une sortie de 128 dimensions. La troisième couche `self.conv3 = GCNConv(128, 3)` prend en entrée des caractéristiques de 128 dimensions et produit une sortie de 3 dimensions.
- Couche linéaire pour la classification finale : Cette couche est utilisée pour effectuer la classification finale du modèle. Elle prend en entrée des caractéristiques de 3 dimensions et produit une sortie de 10 dimensions. Elle est définie par `self.fc1 = nn.Linear(3, 10)`.

Dans la méthode ‘forward’ , les couches sont appliquées séquentiellement avec des activations ReLU entre chaque couche de convolution. Finalement, une

opération de pooling spatial en moyenne est effectuée, suivie d'un aplatissement de la sortie pour l'entrée de la couche linéaire. La fonction de softmax est appliquée sur les logits pour obtenir des probabilités de classe.

L'architecture globale du modèle peut être résumée comme suit : [En-

```
MyGraphModel(
    (linear): Linear(in_features=32, out_features=3, bias=True)
    (conv1): GCNConv(3, 64)
    (conv2): GCNConv(64, 128)
    (conv3): GCNConv(128, 3)
    (fc1): Linear(in_features=3, out_features=10, bias=True)
)
```

FIGURE 5.15 – L'architecture générale du modèle GNN

tré] -> [Couche linéaire] -> [GCN (conv1) -> ReLU] -> [GCN (conv2) -> ReLU] -> [GCN (conv3) -> ReLU] -> [Pooling] -> [Aplatissement] -> [Couche linéaire (fc1)] -> [Softmax] -> [Sortie]

5.4.3 Méthodes d'entraînement

Dans Ce code, nous effectuons l'entraînement d'un modèle GNN (Graph Neural Network) sur un jeu de données spécifique.

Tout d'abord, nous définissons le nombre de noeuds dans le graphe à l'aide de la variable num_nodes. Cette valeur est utilisée pour configurer le modèle en fonction de la taille du graphe. Ensuite, nous instancions le modèle GNN en utilisant la classe MyGraphModel et en passant le nombre de noeuds comme argument. Cela crée une instance spécifique du modèle adaptée au problème en question. Pour représenter les relations entre les noeuds du graphe, nous créons une matrice appelée edge_index. Cette matrice est initialisée avec des zéros et remplie avec les indices appropriés en parcourant les noeuds. Cela établit les connexions entre les noeuds du graphe. Nous définissons le taux d'apprentissage (learning_rate) et le nombre d'époques (num_epochs). Le taux d'apprentissage contrôle la vitesse à laquelle les paramètres du modèle sont mis à jour lors de l'entraînement, et le nombre d'époques indique combien de fois le jeu de données sera parcouru pendant l'entraînement.

En fonction de la disponibilité du GPU, nous déterminons le dispositif d'exécution. Si un GPU est disponible, le modèle est déplacé sur le GPU pour accélérer les calculs. Sinon, le modèle reste sur le CPU par défaut. Nous définissons la fonction de perte en utilisant nn.CrossEntropyLoss(), qui mesure l'écart entre les prédictions du modèle et les étiquettes réelles du jeu de données. De plus, nous initialisons l'optimiseur avec torch.optim.Adam() pour mettre à jour les paramètres du modèle en utilisant la méthode d'optimisation Adam. Nous créons des listes vides pour stocker l'historique de la perte d'entraînement, de l'exactitude d'entraînement et de l'exactitude de test. Ces listes seront

utilisées pour enregistrer les valeurs correspondantes à chaque époque et tracer les courbes de performance. Dans la boucle principale d'entraînement, nous parcourons chaque époque. À chaque itération de la boucle, nous chargeons les images et les étiquettes du jeu de données d'entraînement à partir du trainloader. Les images et les étiquettes sont transférées sur le dispositif d'exécution (GPU ou CPU) en fonction de leur disponibilité. Cela nous permet de profiter de l'accélération GPU si elle est disponible. Nous effectuons une passe avant à travers le modèle en utilisant les images et les indices d'arêtes (edge_index) pour obtenir les prédictions du modèle, appelées logits.

Ensuite, nous calculons la perte en utilisant la fonction de perte et effectuons la rétropropagation pour calculer les gradients et mettre à jour les paramètres du modèle à l'aide de l'optimiseur. Les valeurs de la perte d'entraînement et de l'exactitude sont mises à jour en fonction des résultats de l'itération. À la fin de chaque époque, nous évaluons les performances du modèle sur l'ensemble de test (testloader). Nous chargeons les images de test, les transférons sur le dispositif d'exécution et obtenons les prédictions du modèle en effectuant une passe avant. Enfin, nous calculons l'exactitude du modèle sur les données de test en comparant les prédictions avec les étiquettes réelles. Les valeurs d'exactitude d'entraînement et de test sont enregistrées dans leurs listes respectives pour une utilisation ultérieure lors de l'affichage des résultats. Après chaque époque, nous affichons les résultats tels que la perte d'entraînement et l'exactitude d'entraînement.

Enfin, nous affichons les performances du modèle sur l'ensemble de test pour évaluer ses capacités de classification d'images.

5.4.4 Résultats obtenue

Après l'entraînement on a obtenu les résultats suivantes :

Le modèle GNN utilisé pour la classification des images sur le dataset CIFAR-10 a montré une progression significative au fil de l'entraînement. La perte a diminué et l'exactitude sur l'ensemble d'entraînement a augmenté de 53.33% à 91.22% après cinq époques. Sur l'ensemble de test, l'exactitude a également augmenté de 62.19% à 88.93%. Ces résultats indiquent que le modèle est capable d'apprendre à différencier efficacement les différentes classes d'images et de généraliser ses prédictions sur de nouvelles données, démontrant ainsi sa performance prometteuse pour la classification des images avec le dataset CIFAR-10.

```
Epoch: 0 | Loss: 1.5891 | Train Accuracy: 53.33%
Test Accuracy: 62.19%
Epoch: 1 | Loss: 1.5695 | Train Accuracy: 68.80%
Test Accuracy: 70.69%
Epoch: 2 | Loss: 1.5512 | Train Accuracy: 74.62%
Test Accuracy: 72.65%
Epoch: 3 | Loss: 1.5386 | Train Accuracy: 83.83%
Test Accuracy: 83.41%
Epoch: 4 | Loss: 1.5295 | Train Accuracy: 91.22%
Test Accuracy: 88.93%
```

FIGURE 5.16 – Résultats de l'entraînement

Pour visualiser les résultats on a utilisé la bibliothèque matplotlib

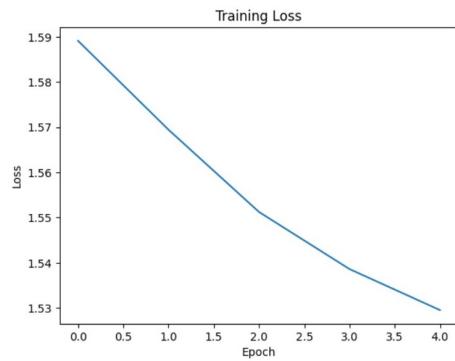


FIGURE 5.17 – Training Loss GNN

Pour tester ces résultats, on a utilisé la fonction "predict-image" :

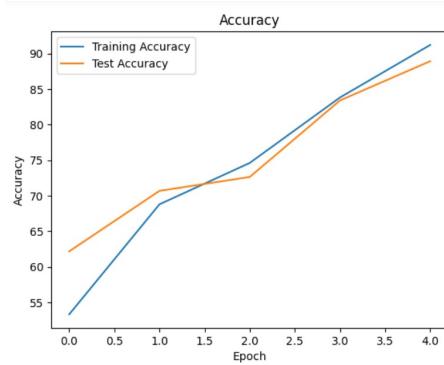
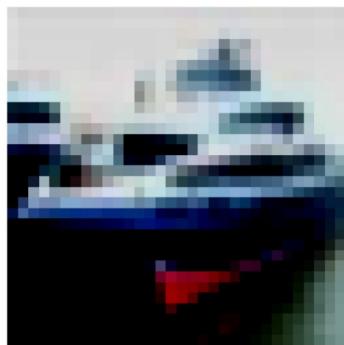


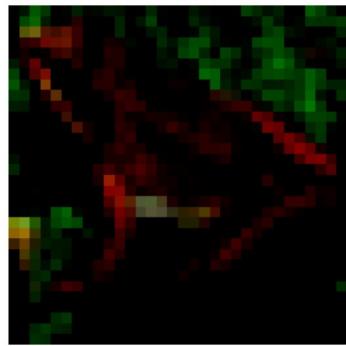
FIGURE 5.18 – test et training accuracy



True Label: ship
Predicted Label: ship

FIGURE 5.19 – Résultat (classe "ship")

En conclusion, grâce à la fonction "predict-image" et à notre modèle GNN, nous avons pu obtenir des prédictions précises sur des images individuelles. Ces résultats démontrent l'efficacité des réseaux de neurones à graphes dans l'analyse d'images et les tâches de vision par ordinateur.



True Label: frog
Predicted Label: frog

FIGURE 5.20 – Résultat (classe "frog")

Conclusion

D'après notre étude et l'implémentation des deux modèles, nous avons obtenu des résultats significatifs en termes de performances de classification pour chaque modèle.

Pour le modèle CNN, nous avons utilisé le dataset MNIST et avons atteint une précision impressionnante de 98%. Le dataset MNIST, qui contient des images en noir et blanc de chiffres manuscrits, est bien adapté aux CNN, où les motifs locaux jouent un rôle crucial dans la classification.

En ce qui concerne le modèle GNN, nous avons utilisé le dataset CIFAR-10 et avons obtenu une précision de 88%. Le dataset CIFAR-10 contient des images couleur de 10 classes différentes, ce qui représente un défi plus complexe pour les GNN en raison de la nature des relations entre les pixels et les régions d'intérêt.

Cependant, il est important de noter que la précision obtenue pour le modèle GNN sur le dataset CIFAR-10 est inférieure à celle du modèle CNN sur le dataset MNIST. Cela indique que le modèle GNN a rencontré plus de difficultés dans la classification des images couleur complexes par rapport aux motifs locaux plus clairs du dataset MNIST.

Ces résultats soulignent l'importance de choisir le modèle approprié en fonction du type d'images, des caractéristiques des données et des performances spécifiques recherchées. Dans notre étude, le modèle CNN a montré une précision plus élevée pour le dataset MNIST, tandis que le modèle GNN a obtenu une précision satisfaisante mais légèrement inférieure sur le dataset CIFAR-10.

En conclusion, notre étude et l'implémentation des deux modèles nous ont permis de constater que le CNN a obtenu une précision de 98% sur le dataset MNIST, tandis que le GNN a atteint une précision de 88% sur le dataset CIFAR-10. Ces résultats mettent en évidence les forces et les limites de chaque modèle dans la classification d'images et soulignent l'importance de choisir le modèle approprié en fonction du contexte et des exigences spécifiques du projet.

Bibliographie

- [1] Les formats d'images numériques. Cours en ligne. France :www.imedias.pro. [En ligne]. Disponible sur : <https://www.imedias.pro/cours-en-ligne/graphisme-design/images-destinees-a-internet/formats-images-numeriques/>
- [2] Team ISTEX. Enrichissements. ISTEX. France : Team ISTEX, (9 juin 2017). [En ligne]. Disponible sur : <https://www.istex.fr/ocr-pre-traitements-des-documents/>
- [3] FUTURA. "Intelligence artificielle : qu'est-ce que c'est ?" FUTURA. France. Le 13 août 2020. [En ligne]. Disponible sur : <https://www.futura-sciences.com/tech/definitions/informatique-intelligence-artificielle-555/>
- [4] Vonintsoa. "Classification d'image : le guide complet." intelligence-artificielle.com, (06 Décembre 2021. France). Édité par Vonintsoa. URL : <https://intelligence-artificielle.com/classification-d-image-guide-complet/>
- [5] Quelle différence entre Machine Learning et Deep Learning? DataScience. France : Team rédac. [En ligne]. Disponible sur : <https://datascientest.com/quelle-difference-entre-le-machine-learning-et-deep-learning>
- [6] Toyoizumi, T. Neural Networks. ScienceDirect. France. Juillet 2023. [En ligne]. Disponible sur : <https://www.sciencedirect.com/journal/neural-networks>
- [7] esri. "Vue d'ensemble de la classification des images." esri. France. [En ligne]. Disponible sur : <https://pro.arcgis.com/fr/pro-app/latest/help/analysis/image-analyst/overview-of-image-classification.htm>
- [8] Pascal Monasse et kimia Nadjahi, lassifiez les images à l'aide de réseaux de neurones convolutifs ;<https://openclassrooms.com/fr/courses/4470531->

classez-et-segmentez-des-donnees-visuelles/5082166-quest-ce-qu-un-reseau-de-neurones-convolutif-ou-cnn

- [9] GEEKFLARE. "Les réseaux de neurones graphiques." GEEKFLARE. France. 24 Janvier 2023. [En ligne]. Disponible sur : <https://geekflare.com/fr/graph-neural-networks/>
- [10] van Rossum, Guido. "Python." Apparu en 1990. Développeurs : Python Software Foundation. Dernière version : 3.1.2. Version en développement : [+/-]. Paradigmes : Objet, impératif. Typage : Fort, dynamique. Influencé par : ABC, C, ICON, Modula-3, Perl, Smalltalk, Tcl. A influencé : Ruby, Groovy, Boo. Implémentations : CPython, Jython, IronPython, PyPy. Licence : Python Software Foundation License. Site Web : www.python.org.