REPUBLIQUE DU CAMEROUN
Paix – Travail – Patrie
- . - . - . -
UNIVERSITÉ DE YAOUNDÉ I
Faculté des Sciences
Département d'Informatique
B.P. 812 Yaoundé

REPUBLIC OF CAMEROON
Peace – Work – Fatherland
- . - . - . -
UNIVERSITY OF YAOUNDÉ I
Faculty of Sciences
Department of Computer Science
P.O. Box 812 Yaoundé

Mémoire en vue de l'obtention du diplôme de
Master en Informatique Fondamentale

Spécialité : **SCIENCE DE DONNÉES**

DETECTION DU PALUDISME EN UTILISANT LES RESEAUX DE NEURONES CONVOLUTIFS PARALLELES

Rédigé et soutenu le 22 février 2024 par :

GUEBOU BOUYIM Ghislaine **17R2256**

Devant le jury constitué de

Président	Examinateur	Rapporteurs
Pr AYISSI Raoul	**Pr TSOPZE Norbert**	**Dr KOUOKAM Etienne** **Dr ATEMEZING Ghislain**

Année académique : 2022/2023

DÉDICACE

À ma maman YAMBI Elise Rosine

REMERCIEMENTS

Nous remercions L'ETERNEL DIEU TOUT PUISSANT qui, dans son amour infini a accordé santé, force, intelligence, sagesse et courage tout au long de ce travail.

Je tiens à témoigner toute ma reconnaissance à mes encadreurs Dr **KOUOKAM Etienne** et Dr **ATEMEZING Ghislain** pour leurs conseils, présences et accompagnements durant toutes les étapes qui ont menées à l'obtention de ce mémoire.

J'exprime également ma profonde gratitude à mes encadrants Dr **MESSI NGUELE Thomas** et Dr **NZEKON Armel** pour leurs soutiens multi-formes qui ont contribué énormément à la réalisations de ce travail et à la construction de mon intellecte.

Je remercie énormément les membres du **Jury** qui ont consacré de leur temps pour l'evaluation de ce manuscrit.

Je remercie également tous les **enseignants du département d' informatique** et de toute l'université de Yaoundé 1 pour leur disponibilité et leur conseil tout au long de ma formation depuis mon cycle licence.

Merci également aux équipes **UMMISCO et HIPERDAS** pour ce confortable cadre de travail mis à notre disposition.

Je tiens également à dire merci à tous mes camarades de promotion à l'instar de Ivan BOUH, Donald ONANA, Mélisa GIAMBOP, Ahmed NSANGOU, Brenda MASSO, Audrey FONGUE, Dorian TCHUENTE, Leandra MAKAMTE, Francis NOAH, Beril EKWELE, Sturm KENFACK pour tous les encouragements et les conseils qu'ils m'ont donnés, ainsi que pour les nombreuses discussions parfois au-delà de la recherche, mais toujours enrichissantes.

Merci à mes parents Monsieur & Madame BOUYIM, à mes frères Landel et Armel, à mes soeurs Murielle et Gilles-isabel, mes tantes, mes oncles, mes cousins et cousines qui m'ont toujours soutenus et encouragés avec amour tout au long de mon parcours.

Merci à tous les intervenants et toutes les personnes qui par leurs paroles, leurs écrits, leurs conseils et leurs critiques ont guidé mes réflexions et ont accepté de me rencontrer et de répondre à mes questions durant mes recherches.

RÉSUMÉ

Le paludisme est une maladie mortelle causant de nombreux décès dans le monde particulièrement en Asie du sud et en Afrique. La manière traditionnelle de diagnostiquer le paludisme consiste à examiner schématiquement des frottis sanguins d'êtres humains à la recherche de globules rouges infectés par des parasites au microscope par un laboratoire ou des techniciens qualifiés. Malheureusement, cette tache est fastidieuse et dépend de l'expérience du personnel médical. Des algorithmes de Deep Learning ont déjà été appliqués aux frottis sanguins du paludisme afin d'améliorer le diagnostic. Mais, l'une des particularités de ces algorithmes est qu'ils sont d'autant plus performant que la quantité de données utilisées est grande. Toutefois, l'exécution séquentielle de ces algorithmes sur des grandes quantités de données est confrontés à de grands défis notamment le temps d'exécution qui peut être très long et le modèle peut être trop volumineux pour la mémoire. Dans ce mémoire nous nous servons de l'algorithme des réseaux de neurones convolutifs (CNN) pour effectuer la détection du paludisme. Nous avons pour cela utilisé un jeu de données d'images de frottis sanguins observé depuis un microscope. Pour avoir à la fois de bonnes performances métriques et un temps d'exécution réduit avec la version séquentielle, nous avons procédé à une implémentation parallèle de l'algorithme d'entraînement. Nous avons proposé une implémentation parallèle basée sur le parallélisme de données sur la partie convolutive car la convolution est l'opération qui a la complexité la plus élevée et prend donc plus de temps lors de l'exécution de l'algorithme par rapport à une stratégie hybride qui combine le parallélisme de données sur la partie convolutive et le parallélisme de modèle sur la couche entièrement connectée. Ce qui justifie les différences de résultats entre nous et cette stratégie qui a obtenu un speedup de 6,16. Les résultats expérimentaux sur une machine de 8 cœurs à 1,80 GHz montrent qu'on obtient de résultats prométteurs avec la stratégie de parallélisation qui a été proposée. De ce fait, nous avons obtenu un speedup de 2,085 et 0,5 pour l'accuracy.

Mots clés : Réseaux de Neurones Convolutifs, deep learning, programmation parallèle, paludisme.

ABSTRACT

Malaria is a deadly disease causing many deaths around the world, particularly in South Asia and Africa. The traditional way of diagnosing malaria is by schematically examining human blood smears for parasite-infected red blood cells under a microscope by a laboratory or trained technicians. Unfortunately, this task is tedious and depends on the experience of the medical staff. Deep learning algorithms have already been applied to malaria blood smears to improve diagnosis. But one of the particularities of these algorithms is that they are more efficient as the quantity of data used is greater. However, the sequential execution of these algorithms on large quantities of data faces great challenges including the execution time which can be very long and the model can be too large for the memory. In this paper we use the convolutional neural networks (CNN) algorithm to perform malaria detection. To do this, we used a dataset of blood smear images observed from a microscope. To have both good metric performance and reduced execution time with the sequential version, we carried out a parallel implementation of the training algorithm. We proposed a parallel implementation based on data parallelism on the convolutional part because convolution is the operation that has the highest complexity and therefore takes more time when executing the algorithm compared to a strategy hybrid that combines data parallelism on the convolutional part and model parallelism on the fully connected layer. Which justifies the differences in results between us and this strategy which obtained a speedup of 6.16. Experimental results on an 8-core machine at 1.80 GHz show that promising results are obtained with the parallelization strategy that was proposed. As a result, we obtained a speedup of 2.085 and 0.5 for accuracy.

Keywords : Convolutional Neural Networks, deep learning, parallel programming, malaria.

Table des matières

Table des figures

Liste des tableaux

Liste des abréviations

CIFAR-10 : Canadian Institute For Advanced Research

CNN : Convolutional Neural Network

CPU : Central Processing Unit

CRISP-DM : Cross Industry Standard Process for Dataming

GPU : Graphic Processing Unit

IA : Intelligence Artificielle

PC : Personal Computer **ReLU** : Rectified Linear Unit

RVB : Rouge Vert Bleu

SGD : Stochastic Gradient Descent

S-PSGD : Synchronous Parallel Stochastic Gradient Descent

.

Chapitre 1

Introduction

Avec les nombreux progrès que connaient le *Deep Learning* aujourd'hui dans plusieurs domaines notamment la santé, il devient donc possible d'améliorer le diagnostic de la maladie mortelle du paludisme comme nous le faisons dans ce mémoire. Nous présenterons dans ce chapitre le contexte, ainsi qu'une idée de la solution au problème que nous voulons résoudre.

1.1 Contexte

Le paludisme est une maladie répandue qui a coûté la vie à des millions de personnes dans le monde entier. Selon l'Organisation mondiale de la santé, environ 627 000 décès résultent de 241 millions d'infections en 2021 [11] . De plus, les régions endémiques où la maladie est répandue comprennent l'Afrique et l'Asie du Sud-Est car dans ces régions du monde et dans d'autres où la mortalité due au paludisme est importante, les ressources nécessaires telles que la prévention fiable, les soins de santé et l'hygiène sont loin d'être adéquates [22] . Dans la plupart des cas, la seule méthode disponible de diagnostic du paludisme est l'examen manuel de la lame microscopique [16]. Malheureusement, les ressources humaines spécialisées sont très souvent limitées dans les zones rurales où le paludisme a une prédominance marquée. Aussi, la microscopie manuelle est subjective et souffre d'un manque de standardisation. Ainsi, de nombreux travaux comme celui effectué dans ce mémoire sont menés dans le but de détecter plus rapidement le paludisme et à réduire le travail manuel.

1.2 Problème

Le *Deep Learning* est un sous domaine du Machine Learning qui utilise comme modèle les réseaux de neurones (ensemble de fonctions connectées entre elles). De plus, plus ils sont profonds, plus ils peuvent réaliser des taches complexes s'ils disposent d'un ensemble de données suffisament grand sur lequel ils peuvent apprendre. Donc, plus la quantité de données utilisées est importante, plus l'algorithme est performant. En contrepartie, l'exécution séquentielle de ces algorithmes sur ces grandes quantités de données peut prendre un temps d'exécution très élevé [6] ou causer un dysfonctionnement de la machine si cette dernière n'est pas assez performante [8]. Il est possible grâce à un usage approprié des architectures multi-cœurs de réduire ce temps d'exécution. Dans nos travaux, nous nous intéressons à la détection du paludisme grace aux images de lame sanguine de patients en utilisant les algorithmes de deep learning. L'objectif est de prendre avantage des architectures matériels multi-cœurs pour réduire le temps d'exécution de ces algorithmes tout en préservant leurs performances métriques.

1.3 Solution proposée

Les algorithmes de *Deep Learning* se sont révélés performants dans de nombreuses applications pratiques comme la reconnaissance vocale ou encore la classification d'images. Les réseaux de neurones convolutifs (CNNs), qui font l'objet de ce mémoire, ont l'avantage de pouvoir traiter des données graphiques comme des images. Cet avantage fait d'eux les algorithmes les mieux adaptés pour des tâches comme la classification d'images [20]. Malgré leurs bonnes performances ça reste un challenge de pouvoir utiliser les CNNs en raison des calculs lourds et complexes à effectuer lors de la formation des modèles. Cela peut prendre plusieurs jours pour avoir un modèle CNN précis avec un grand ensemble de données [20]. Nous proposons dans ce mémoire un algorithme d'entraînement parallèle d'un modèle CNN destiné à la prédiction (classification) des images de lames sanguines. L'objectif étant de réduire le temps d'exécution tout en préservant les performances métriques (que l'on obtiendrait lors d'un entraînement séquentiel). Pour ce faire, nous utilisons le parallélisme de données sur la partie convolutive du CNN.

1.4 Plan du mémoire

Dans la suite, nous allons au chapitre 2 présenter les notions nécessaires à la compréhension de nos travaux, et les travaux existant similaires aux nôtres. Ensuite au chapitre

3 nous présenterons l'algorithme de réseaux de neurones convolutifs, ainsi que l'approche utilisée pour l'implémentation parallèle. Nous présenterons au chapitre 4 les résultats expérimentaux actuels, et enfin le chapitre 5 permettra de conclure et de donner quelques perspectives.

Chapitre 2

Généralités et Travaux existants

Dans ce chapitre, nous présenterons tout d'abord le *Machine Learning* de façon globale puis nous verrons plus en détail un de ses algorithmes d'apprentissage à savoir les CNNs et nous aborderons par la suite les concepts liés à la programmation parralèle en deep learning. Nous présenterons ensuite les travaux existants concernant la détection du paludisme avec les CNNs ainsi que sur la parallélisation des CNNs .

2.1 Généralités

Dans cette section, nous présentons les notions liées aux *Machine Learning*, réseaux de neurones convolutifs et la programmatioon parallèle en *Deep Learning*.

2.1.1 Machine Learning

Le *Machine Learning* est une branche de l'intelligence artificielle (IA) qui permet aux systèmes informatiques d'apprendre directement à partir des données et d'expériences [18]. Elle consiste à laisser des algorithmes découvrir des "patterns" , à savoir des motifs récurrents, dans les ensembles de donnée (des chiffres, des mots, des images, des statistique, ...). La mise en oeuvre d'un modèle de Machine Learning repose sur plusieurs étapes comme le montre la figure 2.1 que nous résumons en ces quelques points :

- **Sélectionner et préparer un ensemble de données d'entraînement** : les données seront utilisées pour nourrir le modèle de Machine Learning pour qu'il apprenne à résoudre le problème pour lequel il est conçu. Selon le mode d'apprentissage, les données peuvent **étiquetées** afin d'indiquer au modèle les caractéristiques qu'il devra identifier pour un **apprentissage supervisé** ou **non étiquetées** alors le modèle devra repérer et extraire les caractéristiques récurrentes de

lui-même on parlera donc d'**apprentissage non supervisé**.

- **Sélectionner un algorithme à exécuter** sur l'ensemble de données d'entraînement. Le type d'algorithme à utiliser dépend du type et du volume de données d'entraînement et du type de problème à résoudre.
- **L'entraînement de l'algorithme** : il s'agit d'exécuter l'algorithme avec des paramètres passés aux variables, et les résultats sont comparés avec ceux qu'il aurait du produire. Les poids et biais peuvent ensuite être ajustés pour accroître la précision du résultat.
- **L'utilisation et l'amélioration du modèle** : ici, on utilise le modèle sur de nouvelles données, dont la provenance dépend du problème à résoudre.

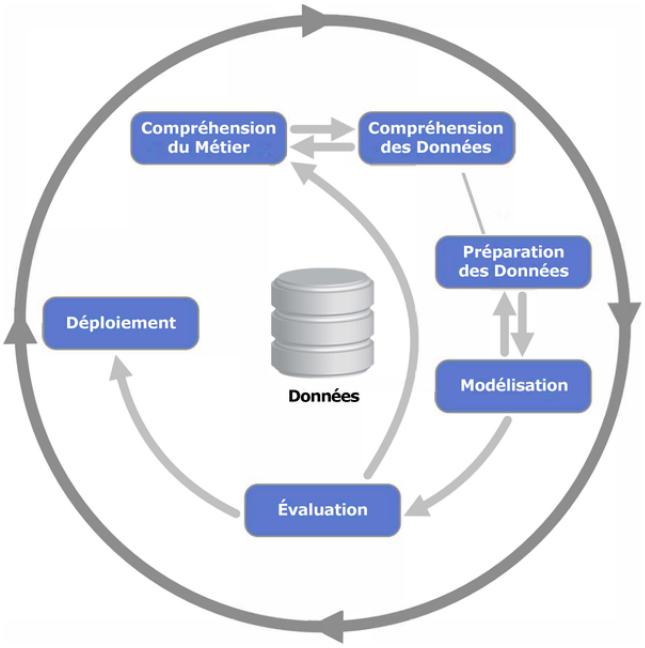

FIGURE 2.1 – *Cross-Industry Standard Process for Data Mining* (CRISP-DM) [23]. Etapes de confection d'un algorithme de machine learning

Il existe une grande variété d'algorithmes de machine learning. Les principaux sont :

- **L'arbre de décision** : Cet algorithme permet d'établir des recommandations basées sur un ensemble de règles de décisions en se basant sur des données classifiées.
- **La regression linéaire** est utilisée pour prédire la valeur d'une variable dépendante basée sur la valeur d'une variable indépendante.
- **Les algorithmes d'association** permettent quant à eux de découvrir des patterns et des relations dans les données, et à identifier les relations » si / alors » appelées » règles d'association «.
- **Les algorithmes de clustering** qui consistent à identifier les groupes présentant

des enregistrements similaires et à étiqueter ces enregistrements en fonction du groupe auquel ils appartiennent.

- **Les réseaux de neurones** sont des algorithmes se présentant sous la forme d'un réseau à plusieurs couches. La première couche permet l'ingestion des données, une ou plusieurs couches cachées tirent des conclusions à partir des données ingérées, et la dernière couche assigne une probabilité à chaque conclusion. Ce sont eux qui constituent le domaine du *Deep Learning* selon leurs multiples variétés à l'instar des CNNs qui constitue l'objet de nos travaux.

2.1.2 Architecture & Fonctionnement des CNNs

Les CNN désignent une sous-catégorie de réseaux de neurones et sont à ce jour un des modèles de classification d'images réputés être les plus performant [5]. Ils sont constitués de 2 principales parties :

- **Une partie convolutive** : Son objectif final est d'extraire des caractéristiques propres à chaque image en les compressant de façon à réduire leur taille initiale. En résumé, l'image fournie en entrée passe à travers une succession de filtres, créant par la même occasion de nouvelles images appelées cartes de convolutions. Enfin, les cartes de convolutions obtenues sont concaténées dans un **vecteur de caractéristiques** appelé code CNN.
- **Une partie classification** : Le code CNN obtenu en sortie de la partie convolutive est fourni en entrée dans une deuxième partie, constituée de couches entièrement connectées appelées perceptron multicouche (MLP pour Multi Layers Perceptron). Le rôle de cette partie est de combiner les caractéristiques du code CNN afin de classer l'image.

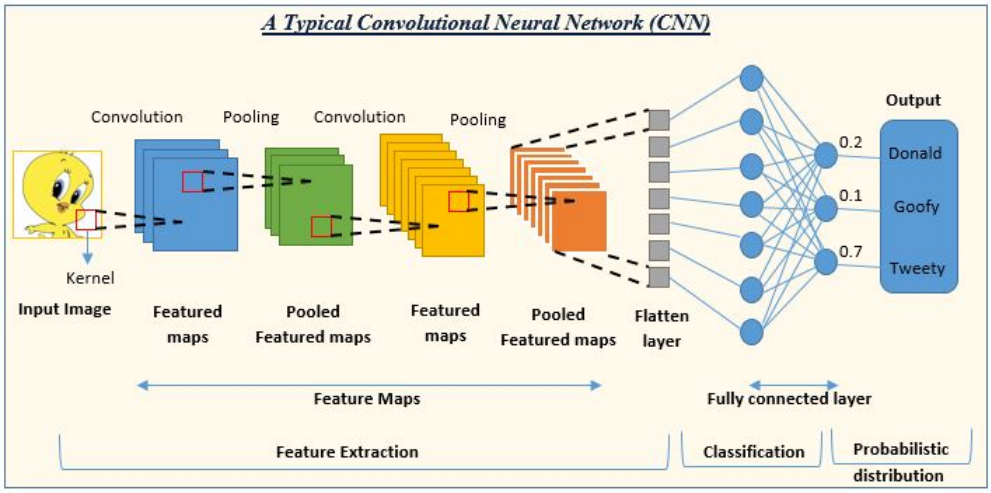

FIGURE 2.2 – Architecture de base d'un CNN [17]

Ces différentes parties possèdent une combinaison de plusieurs couches à savoir :

- **La couche de convolution** : Le rôle de cette première couche est d'analyser les images fournies en entrée et de détecter la présence d'un ensemble de features afin de produire des cartes de caractéristiques grace à l'opération de convolution comme le montre la figure ci-dessous.

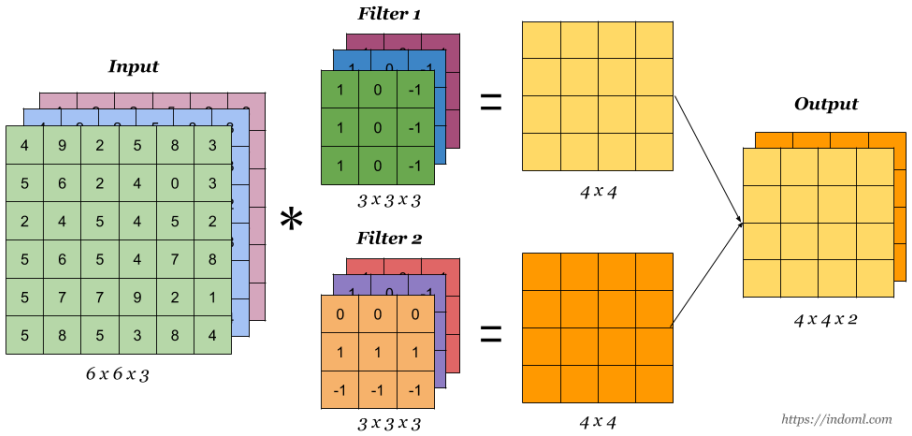

FIGURE 2.3 – Opération de convolution sur une image dans une couche à 2 filtres [9]

L'opération de convolution s'effectue entre une image (matrice de nombres positifs) et un filtre (matrice carrée de nombre) suivant leur nombre de canal, du pas s (saut fait pendant le processus de calcul) et du padding p (p=0 pour un padding valid : la taille de la carte de caractéristique est différente de l'image d'entrée ; p=1 pour un padding meme : la taille de l'image convoluée est la meme que l'image d'entrée). La convolution s'effectue ainsi :

$$\underbrace{(I * F)(i,j) = (F * I)(i,j)}_{\text{commutative}} = \sum_m \sum_n F(m,n) I(i-m, j-n)$$

FIGURE 2.4 – Opération de convolution entre une image et un filtre [9]

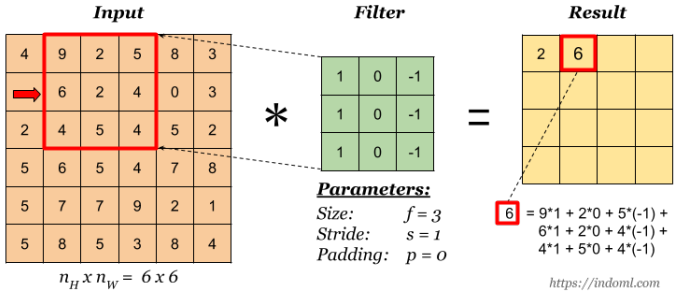

FIGURE 2.5 – Opération de convolution entre une image et un filtre [9]

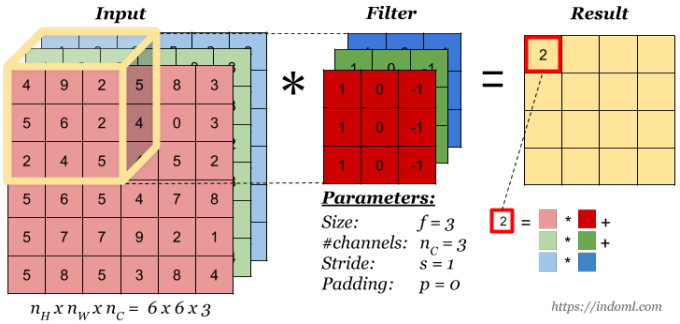

FIGURE 2.6 – Opération de convolution entre une image et un filtre constitués de plusieurs canaux [9]

- **La couche de mise en commun (Pooling)** : La couche de Pooling est une opération généralement appliquée entre deux couches de convolution. Celle-ci reçoit en entrée les features maps formées en sortie de la couche de convolution et son rôle est de réduire la taille des images, tout en préservant leurs caractéristiques les plus essentielles. Parmi les plus utilisés, on retrouve le max-pooling dont l'opération consiste à conserver à chaque pas, la valeur maximale de la fenêtre de filtre ou encore l'average pooling dont l'opération consiste à conserver à chaque pas, la valeur moyenne de la fenêtre de filtre comme l' illustre la figure 2.13.

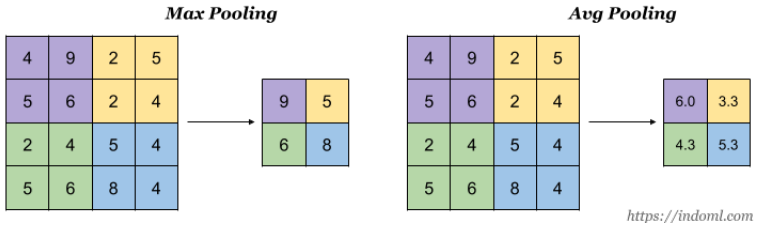

FIGURE 2.7 – Max pooling & Average pooling [9]

Suivant cette figure, pour obtenir une entrée de notre carte de caractéristique. On peut proceder ainsi :

— Max pooling : x = $\max_{i,j=1,\ldots,N} I_i, j$

— Average pooling : x = $1/N(\sum_{i=1}^{N} I_i)$

- **La couche entièrement connectée** : c'est un ensemble de couches cachées constituées de neurones connectés les uns aux autres et placées en fin d'architecture de CNN. Après avoir reçu un vecteur en entrée, chaque neurone applique successivement une combinaison linéaire puis une fonction d'activation dans le but final de classifier l'image d'entrée (voir schéma suivant). Elle renvoie enfin en sortie un vecteur de taille correspondant au nombre de classes dans lequel chaque composante représente la probabilité pour l' image d'entrée d'appartenir à une classe.

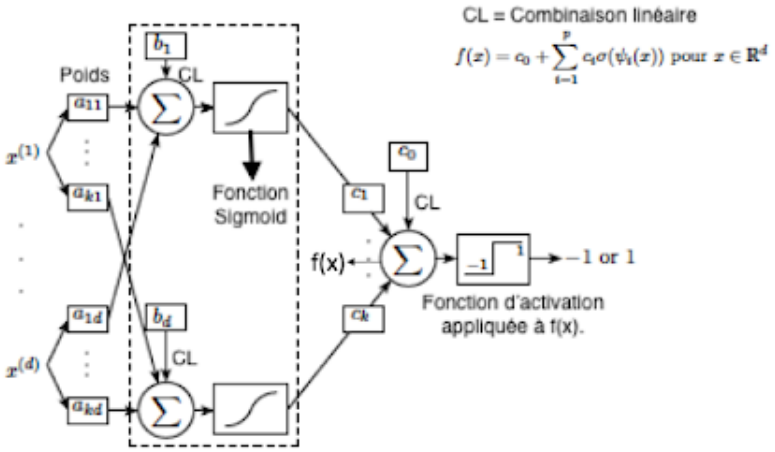

FIGURE 2.8 – Fonctionnement d'un réseau neuronal à 2 couches cachées [5]

2.1.3 Programmation parallèle en deep learning

Le paradigme de programmation parallèle désigne l'ensemble de méthodes permettant de prendre avantages des architectures multi-cœurs pour écrire des programmes avec des

instructions s'exécutant simultanément, afin de réduire le temps d'exécution. La possibilité d'écrire du programme parallèle en deep learning dépend des ressources matérielles, la complexité du modèle et la dimension des données ainsi que leur quantité. Selon le cas de figure, le choix se pose généralement sur le modèle de programmation parallèle et le type de parallélisme.

Modèles de programmation parallèle

Un modèle de programmation parallèle désigne la manière avec laquelle un algorithme parallèle sera implémenté sur une architecture particulière. Il est principalement constitué de deux éléments, à savoir l'organisation de la mémoire et la décomposition du problème. L'organisation de la mémoire désigne la façon dont communiquent les différentes unités de traitement, on distingue principalement 02 types d'organisations à savoir :

- **Organisation à mémoire distribuée** : cette organisation correspond à un ensemble d'unités de traitement appelé noeuds, et d'un réseau d'interconnexion entre les noeuds ainsi qu'un support de transfert de données [14]. Comme le montre la figure 2.9 où, chaque noeud est indépendant des autres et est constitué de son processeur, sa mémoire locale et parfois de ses périphériques.

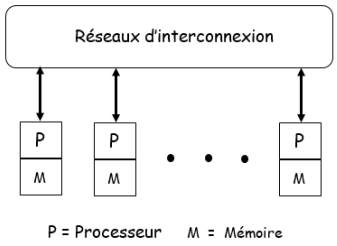

FIGURE 2.9 – organisation à mémoire distribuée

- **Organisation à mémoire partagée** : C'est l'organisation mémoire que nous avons utilisée tout au long de nos travaux. Elle désigne une machine physique avec plusieurs processeurs ou cœurs et une mémoire physique partagée (mémoire globale), ainsi qu'un réseau d'interconnexion pouvant connecter les processeurs à la mémoire (voir figure 2.14). Les cœurs d'un processeur multicœur sont un exemple pour cette organisation mémoire.

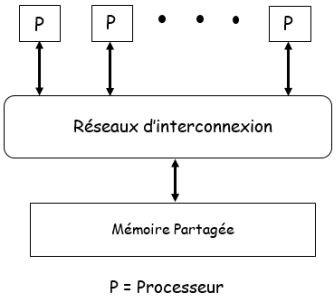

FIGURE 2.10 – organisation à mémoire partagée

La décomposition du problème désigne le niveau de parallélisation que l'on désire implémenter en fonction des calculs effectués dans la version séquentielle d'un algorithme. Il existe principalement 04 niveaux de parallélisation [14] à savoir : La parallélisation au niveau des instructions, au niveau des tâches, au niveau des boucles et au niveau des données. Cette dernière (au niveau des données) considérée dans ce mémoire, consiste à diviser les données initiales en plusieurs sous-ensemble. Chaque sous-ensemble est ensuite attribué à une unité de traitement pour une exécution simultanée.

Les types de parallélisme en deep learning

Les réseaux de neurones profonds sont efficaces pour extraire les caractéristiques significatives et modéliser les données pour des tâches données [21]. Parfois, lorsque les données sont de grande dimension ou que le nombre de paramètres dans le modèle est très élevé, nous devons alors effectuer des calculs élevés. Dans de tels cas, l'apprentissage profond parallèle ou distribué devient utile pour réduire l'effort nécessaire aux calculs élevés. Pour ce faire, Il existe deux principaux types de parallélisme :

- **Le parallélisme des données** peut être défini comme la division des données en N partitions où chacune des partitions peut être utilisée pour l'entraînement sur différentes machines ou périphériques comme des cœurs de processeur, des GPU ou même des machines comme illustré ci-dessous.

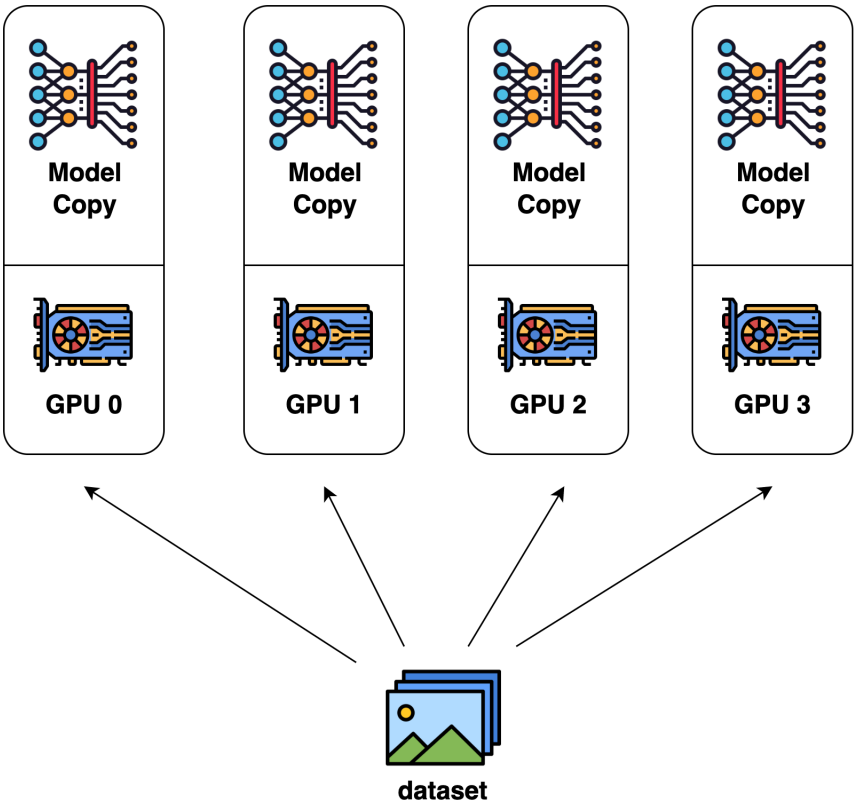

FIGURE 2.11 – Parallélisme des données [19]

En parlant de la manière traditionnelle, où la procédure de formation produit un gradient pour chaque mini-lot, après avoir appliqué le parallélisme des données, la question est maintenant de savoir comment ces N gradients (∇P) doivent être combinés. Selon le besoin on enregistre 2 méthodes :

SGD synchrone : il s'agit du processus dans lequel la moyenne du gradient N est utilisée pour mettre à jour une fois les paramètres du modèle. L'utilisation de la moyenne peut produire un gradient précis. L'inconvénient majeur de cette technique est qu'elle nécessite de terminer tous les calculs dans tous les cœurs pour son gradient (∇P) local [21].

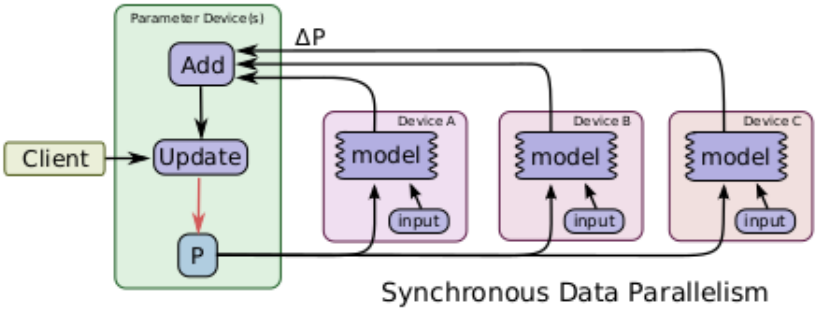

FIGURE 2.12 – SGD Parallèle synchrone

SGD asynchrone : dans ce processus, tous les gradients (∇P) sont utilisés pour mettre à jour les paramètres du modèle sans les combiner. En utilisant ce processus, nous pouvons avoir N mises à jour de paramètres du modèle.

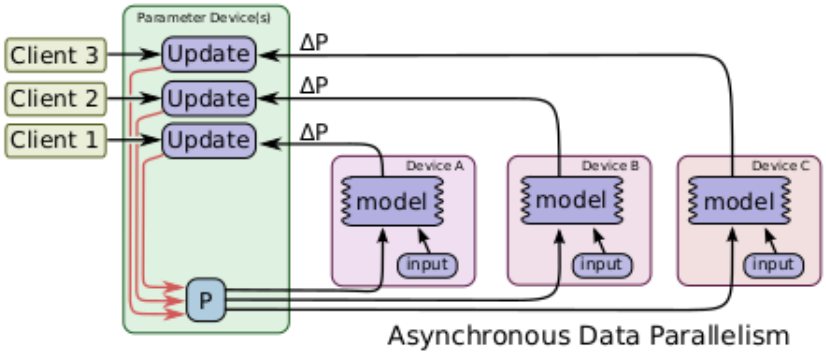

FIGURE 2.13 – SGD Parallèle Asynchrone

- **Parallélisme du modèle** divise un modèle (c'est-à-dire ses couches ou ses tenseurs) sur plusieurs cœurs, contrairement au parallélisme des données, répliquant le même modèle pour tous les cœurs de formation. Suivant le besoin, on distingue 2 façon de faire :

  Le parallélisme des pipelines : le modèle est divisé par couche en plusieurs morceaux, chaque morceau étant attribué à un périphérique. Lors du passage en avant, chaque appareil transmet l'activation intermédiaire à l'étape suivante. Lors du passage en arrière, chaque appareil renvoie le gradient du tenseur d'entrée à l'étage de pipeline précédent. Cela permet aux appareils de calculer simultanément et augmente le débit de formation. L'un des inconvénients de la formation parallèle sur pipeline est qu'il y aura un certain temps de bulle pendant lequel certains appareils sont engagés dans le calcul, ce qui entraînera un gaspillage de ressources de calcul [19].

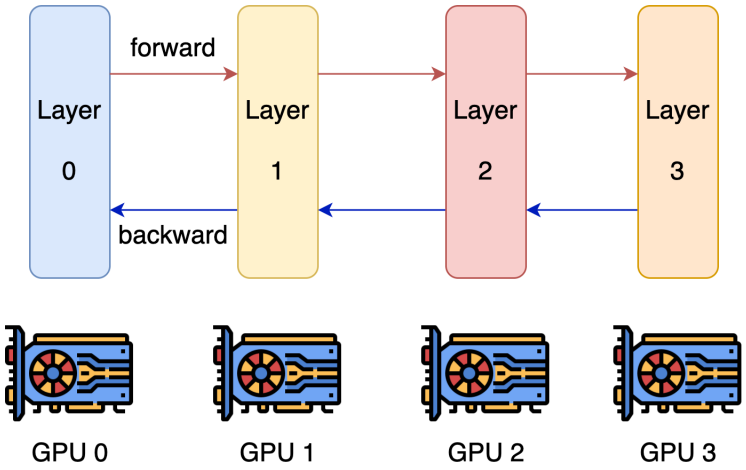

FIGURE 2.14 – Parallélisme de pipeline [19]

Le parallélisme tensoriel consiste à paralléliser le calcul dans une opération telle que la multiplication matrice-matrice. Cela revient à diviser un tenseur en N morceaux le long d'une dimension spécifique et chaque appareil ne contient 1/N du tenseur entier sans affecter l'exactitude du graphe de calcul. Cela nécessite une communication supplémentaire pour s'assurer que le résultat est correct.

2.2 Travaux existants

Cette section est dédiée aux travaux existants similaires aux nôtres. Nous présenterons séparément les travaux antérieurs sur l'usage du deep learning pour la détection du paludisme, et d'autre part les travaux antérieurs sur la parallélisation des algorithmes de deep learning.

2.2.1 Détection du paludisme avec le deep learning

La détection ou prédiction du paludisme est une tâche de classification supervisée en machine learning (classification d'images). De nombreux travaux ont vu le jour avec pour objectif d'effectuer cette tâche le mieux que possible. Nous présentons ici certains de ces travaux basés sur les algorithmes de *deep learning* en particulier ceux qui utilisent les CNNs.

Feng Yang et al. [7] dans leurs travaux utilisent un CNN pour la détection et du comptage automatique des parasites du paludisme dans des images numériques de frottis

sanguins épais acquis avec des smartphones. Le jeu de données contient au total 1819 images de frottis sanguins proovenant de 150 patients. Ils appliquent au préalable un ensemble d'opérations de pré-traitement au jeu de données, à savoir la segmentation et le débruitage des images afin d'extraire des informations utiles sur les composants sanguins infectés et non infectés. Chaque image couleur est convertie en une image en niveaux de gris par la méthode de seuillage et le débruitage est suivi pour améliorer la qualité de l'image. Ils obtiennent un modèle ayant comme performances métriques, une précision à 78,98%, rappel à 80,81 % et l'accuracy à 97,26% .

Yuhang Dong et al. [22] proposent une architecture basée sur les CNNs pour la détection du paludisme. Pour la classification très précise des cellules infectées par le paludisme à l'aide de réseaux de neurones à convolution profonde. Ils utilisent les méthodes de traitement d'image pour la segmentation des globules rouges à partir d'images entières. Puis ils présentent des procédures de compilation d'un ensemble de données d'images organisées par des pathologistes pour la formation d'un réseau de neurones profonds, ainsi que des méthodes d'augmentation de données utilisées pour augmenter considérablement la taille de l'ensemble de données, à la lumière du problème de surajustement associé à la formation de réseaux de neurones à convolution profonde. L'entraînement est effectué avec un jeu de données contenant 2703 images. Ils obtiennent un modèle ayant comme performances métriques, une précision à 95% .

Ces travaux illustrent l'efficacité que peut avoir un CNN lorsqu'il est utilisé pour une tâche de détection du paludisme. Cependant, les auteurs que nous avons précédemment cité n'ont pas pris en compte le temps pour la formation du modèle. Notre objectif est donc d'utiliser un CNN pour la détection du paludisme, en prenant en compte non seulement les mesures métriques de performance, mais aussi le temps nécessaire pour la phase d'entraînement.

2.2.2 Parallélisation des algorithmes de deep learning

Plusieurs travaux ont été réalisés sur l'implémentation parallèle des algorithmes de deep learning. En analysant ces travaux, l'on observe principalement deux approches de parallélisation à savoir la parallélisation au niveau des données qui peut s'effectuer de façon synchrone ou asynchrone et la parallélisation du modèle (au niveau des instructions) qui peut etre de deux types : le parallélisme de pipeline et le parallélisme de tenseur.

C'est par exemple le cas de Martin Abadi et al. [1] qui, dans leurs travaux, pré-

sentent l'algorithme S-PSGD (*Synchronous Parallel Stochastic Gradient Descent*). C'est une implémentation parallèle de la descente stochastique du gradient exploitant la parallélisation au niveau des données. L'algorithme considère plusieurs unités de traitement, chacune avec une copie locale du modèle global et un sous-ensemble du jeu de données. Chaque unité de traitement calcule les gradients (∇P) des paramètres de son modèle local en utilisant le sous-ensemble du jeu de données qui lui a été attribué. L'algorithme est synchrone au sens où, les unités de traitements se synchronisent entre eux pour agréger leur gradient afin de mettre à jour le modèle global. Ce modèle de parallélisation est plus facile à mettre en place que se soit dans un environnement à mémoire partagée ou distribuée [4]. Malgré quelques désavantages dus à la synchronisation à savoir : le fait que si une unité de traitement tombe en panne, c'est l'ensemble du système qui devient défaillant. Il reste tout de même le modèle de parallélisation le plus utilisé.

Jeffrey Dean et al. [6] combinent la parallélisation du modèle et la parallélisation au niveau des données de façon asynchrone. Ils présentent l'algorithme *DownPour* SGD. Le principe est que, chaque unité de traitement est chargée de mettre à jour un sous-ensemble de paramètres du modèle. Cette approche est asynchrone au sens où les unités de traitement fonctionnent indépendamment les unes des autres. La mise à jour d'un sous-ensemble de paramètres du modèle est effectuée par une unité de traitement dès qu'elle a terminé les calculs des gradients. Cet algorithme a pour avantage d'éliminer le temps nécessaire pour la synchronisation, mais il est plus approprié lorsque le modèle à entraîner est très volumineux (modèle d'apprentissage non supervisé) pour tenir en mémoire sur une unité de traitement. Il ne nécessite qu'une quantité réduite de la mémoire pour chaque appareil, pour stocker et effectuer les traitements liés à un sous-ensemble de paramètres du modèle [12]. Par contre il est moins utilisée que la version synchrone car il est plus difficile à implémenter et la convergence n'est pas stable.

Sunwoo Lee et al. [20] parallélisent les modèles CNN en utilisant le parallélisme des données. Les modèles sont entraînés à l'aide de SGD synchrone de telle sorte que le résultat de l'entraînement parallèle soit exactement le même que le résultat de l'entraînement séquentiel. La stratégie de parallélisation repose sur deux techniques clés pour maximiser le chevauchement des calculs et des communications. Tout d'abord, ils regroupent tous les gradients de paramètres en deux gros morceaux et les réduisent sur tous les nœuds à l'aide de communications asynchrones. Sur la base d'une analyse de dépendance des données, ils maximisent le chevauchement en choisissant la taille optimale de chaque morceau de gradient. Puis, ils reproduisent le calcul du gradient dans quelques couches entièrement connectées. Ils implémentent leur CNN parallèle avec le dataset ImageNet qui contient

1,2 million d'image RVB où ils obtiennent un 62,97x comme speed up sur 128 coeurs.

Chi-Chung Chen et al. [3] présentent une implémentation du parallélisme de modèles pipeline qui permet une utilisation élevée du GPU tout en maintenant une précision d'entraînement robuste via une nouvelle technique de prédiction de poids, SpecTrain. Dans cette méthode, dès les premiers stades du pipeline, le mini-lot prédit la future version des poids (Wct), qui devrait devenir la pondération la plus mise à jour au temps t, et utilise cette version des poids pour effectuer le calcul. Ils implémentent leur approche sur le jeu de donnée CIFAR-10 qui contient 50000 images pour l'entrainement et 10000 pour le test où ils btiennent un speed up de 8,91x sur 4GPU.

Alex Krizhevsky [10] proposent un algorithme parallèle d'entraînement d'un CNN pour la classification d'image pour paralléliser la formation des réseaux convolutifs, ils s'appuient fortement sur le parallélisme des données dans la partie convolutive et sur le parallélisme des modèles dans la partie entièrement connectée. Pour leur implémentation, chaque unité de traitement contient en mémoire une copie complète du modèle, mais aussi un sous ensemble du jeu de données au niveau de la partie convolutive ; une fois au niveau de la couche entièrement connectée les unités de traitement travaillent de façon collégiale sur chaque couche cachée sur un meme lot de donnée. Cette implémentation a pour avantage de réduire le temps de communication entre les unités de traitement, mais par contre le choix de fonction ou méthode d'agrégation est crucial pour garantir une bonne convergence.

Le point commun des travaux que nous venons de citer est le fait qu'ils utilisent tous la descente de gradient stochastique et majoritairement des GPUs comme unité de traitement. La principale différence est faite sur la stratégie utilisée pour la mise à jour du modèle global. Nous notons aussi que le parallélisme de donnée est le plus utilisé et que le parallélisme de modèle dispose de certains manquement comme l'apparition d'un temps *bulle* pendant lequel certains appareils ou unité de traitement ne sont pas engagés dans le calcul, ce qui entraînera un gaspillage de ressources de calcul [19] et le problème d'obsolescence des poids, c'est-à-dire que les gradients sont calculés à un moment donné avec des poids obsolètes, ce qui conduit à instabilité de l'entrainement et une perte de précision [3]. Le tableau 2.1 présente un récapitulatif des travaux existants dont nous avons discutés, ainsi que notre positionnement par rapport à ces travaux.

Articles	**Objectif Principal**	**Stratégie de parallélisation**	**Algorithme utilisé**	**Résultats**
Feng Yang et al. [13] (2019)	Détection du paludisme	Aucune stratégie présenté	CNN	f-mesure = 80,81% ; accuracy = 97,26% ; précision = 78,98%
Yuhang Dong et al. [3] (2018)	Détection du paludisme	Aucune stratégie présentée	CNN	précision = 95%
Martin Abadi et al. [1] (2016)	Système distribué pour le machine Learning	Parallélisation des données : S-PSGD (*Synchronous Parallel Stochastic Gradient Descent*)	non spécifié	non spécifié
Jeffrey Dean et al. [6] (2012)	Réduire le temps d'entraînement d'un réseaux de neurone profond	Parallélisation du modèle + Parallélisation des données (Asynchrone)	réseaux de neurone profond (1,7 milliard de paramètres)	SpeedUp = 12 avec 21 unité de traitements
Alex Krizhevsky [10] (2014)	Paralléliser les CNN	Parallélisme des données sur la partie convolutive + Parallélisme du modèle sur la couche entièrement connectée	CNN	SpeedUp = 6,16 avec 8 unités de traitements
Sunwoo Lee et al. [20] (2017)	Paralléliser les CNN qui maximise le chevauchement de la communication inter-processus avec le calcul	Parallélisme des données de façon synchrone	CNN	SpeedUp = 62,97 avec 128 unités de traitements
Chi-Chung Chen et al. [3] (2019)	Paralléliser les CNN	Parallélisation du modèle avec prédiction de la valeur des poids à un instant donnée	CNN	SpeedUp = 8,91 avec 4 unités de traitements
Dans ce mémoire	Réduire le temps d'entraînement d'un CNN	Parallélisme des données sur la partie convolutive	CNN	SpeedUp = 2,085 avec 8 unités de traitements ; précision = 50%

TABLE 2.1 – Récapitulatifs des travaux existants et idée de la solution

Chapitre 3

Parallélisation des CNNs

Les réseaux de neurones convolutifs (CNNs ou ConvNets) sont des types de réseaux de neurones utilisés pour analyser les images afin de reconnaître des objets, des classes ou des catégories. Pour cette raison, ils sont particulièrement adaptés pour plusieurs tâches en vision par ordinateur [2] comme la détection d'obet, la classification d'image, la segmentation d'image ou bien la segmentation des instances. Malgré leurs bonnes performances, le temps d'entraînement pour former un modèle avec un CNN peut être très grand (plusieurs jours). Un moyen intuitif pour accélérer la formation d'un CNN est via la programmation parallèle [3] comme nous le faisons dans ce mémoire. Nous présenterons dans ce chapitre le principe de formation d'un modèle de CNN ainsi que la stratégie d'implémentation parallèle que nous avons proposée.

3.1 Algorithme Séquentiel

La formation des CNNs s'effectue par l'entrainement du modèle sur un jeu de données. Elle se fait itérativement suivant 2 étapes principales à savoir la propagation (forward) et la rétropropagation (backward) comme nous le voyons ci-dessous :

L'algorithme 1 présente l'entraînement séquentiel d'un CNN pour la classification d'image. La procédure d'optimisation utilisée est la descente de gradient version mini batch où l'on met à jour des paramètres Θ du modèle après avoir traité M éléments du jeu de données [15]. Elle se déroule principalement comme suit :

1. On partitionne le jeu de données en plusieurs lots $\mathsf{b_i}$ de taille M (ligne 3)
2. Pour chaque époque[1] (ligne 7 à 25), l'on parcourt tous les lots $\mathsf{b_i}$. Pour chaque

1. **une époque** fait référence à un passage complet de l'ensemble du jeu de données d'entraînement

Algorithm 1 : **Training**

Input (X,Y) : ensemble d'images x et leur label y ; λ : le learning rate ;
M : le batch size ; epoch : le nombre d'époques maximal
Output Θ : Ensemble de paramètres du modèle
Start

```
1: initializeParameters(Θ)            // on initialise les paramètres Θ du modèle
2: (xtrain, ytrain) (xval, yval) ← splitData(X, Y) // on réserve une partie du jeu de données pour la validation
3: B ← getBatch(M,xtrain,ytrain) //on crée des lots de données de taille M
4: e ← 1
5: stop ← 0
6: valLoss ← ∞
7: while e ≤ epoch and stop < 4 do
8:     for b_i ∈ B do
9:         loss ← 0
10:        zeroInitialize(dΘ)          // pour chaque lots de données, on initialise les dérivées partielles à zéro
11:        for (x, y) ∈ b_i do
12:            y_pred ← Forward(x, Θ)  // pour chaque phrase x dans un lot de donnée, l'on prédit sa classe
13:            loss ← loss + lossEntropy(y, y_pred) // on accumule l'erreur de prédiction
14:            dΘ ← dΘ + Backforward(Θ, x, y, y_pred) //on calcule et on accumule les dérivées partielles
15:        end for
16:        update(Θ, dΘ, λ, M)         //on mets à jour les paramètres
17:    end for
18:    if evalModel(Θ, xval, yval) > valLoss then
19:        stop ← stop + 1             // si les performances du modèle n'ont pas augmenté, on incrémente la variable stop
20:    else
21:        valLoss ← evalModel(Θ, xval, yval)
22:        stop ← 0                    // si non, la variable stop est réinitialisé à zéro
23:    end if
24:    e ← e + 1
25: end while
26: Return Θ
End
```

phrase x dans $\mathsf{b_i}$, l'on prédit sa classe (ligne 12). L'on calcule et on accumule l'erreur de prédiction ainsi que les différentes dérivées partielles (ligne 13 et 14).

3. Une fois qu'on a traité un lot $\mathsf{b_i}$, l'on se sert des dérivées partielles précédemment accumulées pour mettre à jour les paramètres (ligne 14).
4. On évalue ensuite le modèle en calculant l'erreur de prédiction sur les données de validations (ligne 18)
5. On arrête l'entraînement si le modèle n'a pas évolué au cours des quatre dernière époques ou alors si l'on a atteint le nombre d'époques maximale (ligne 7).

Comme dit plus haut l'un des points essentiels de l'entrainement est le forward (propagation avant). L'algorithme ci-après présente plus en détail son fonctionnement.

Le second point crutial est la Backforward (rétropropagation) dans laquelle on calcule progressivement les gradients dans les différentes couches de la dernière à la première

à travers l'algorithme.

Algorithm 2 : **Forward**

Input x : une image du jeu de données ;
Θ : les paramètres de la partie entièrement connectée ;
γ : Les paramètres de la partie convolutives
Output y_{pred} : Valeur prédite par le modèle
Start

1: i ← 1
2: **while** i <= nbrecouche **do**
3: x ← Convolution(x,γ)
4: x ← $\mathsf{Fonction}_{\mathsf{activation}}$(x) *// très souvent Relue ou TanH*
5: x ← Pooling(x)
6: i ← i + 1
7: **end while**
8: V ← Vectorisation(x)
9: Z ← Θ .V + b_{FC}
10: y_{pred} ← f(Z) *// f est une fonction d'activation qui peut être selon le cas sigmoïde ou softmax ...*
11: **Return** y_{pred}

End

grace à la propriété mathématique de dérivation en chaîne. L'algorithme ci-après nous décrit le procedé permetant l'obtention de ces gradients.

Algorithm 3 : **Backforward**

Input x : une image du jeu de donnée ; y_{pred} : Valeur prédite par le modèle ;
y_{train} :label de l' image ; Θ : les paramètres de la partie entièrement connectée ;
γ : Les paramètres de la partie convolutive
Output dΘ : Ensemble des dérivées partielles du modèle
Start

1: n ← LENGTH(x)
2: ZeroInitialiZe(dΘ, dγ) *// On initialise les gradients à zéro*
3: $dW \leftarrow \mathsf{Vect}_{\mathsf{image}}^{\mathsf{Transpos}}.(y_{pred} - y_{train})$ *// dW matrice de gradients des poids de la couche entièrement connectée*
4: $db_{FC} \leftarrow \sum(y_{pred} - y_{train})$ *// db_{FC} gradients des biais de la couche entièrement connectée*
5: $dx \leftarrow \mathsf{Convolution}(\mathsf{padded}(dO), \mathsf{filtre}_{\mathsf{roter-de-180}})$ *// dO gradients propagés provenant de la couche précédente*
6: $dF \leftarrow \mathsf{Convolution}(x, dx)$ *// dF gradients des filtres et dx est la dérivée partielle de l'image*
7: $db_C \leftarrow \sum(dO)$ *// db_C gradient des biais de la partie convolutive*
8: **Return** $\{dW, dF, db_{FC}, db_C\}$

End

3.2 Stratégie de Parallélisation

L'approche de parallélisation que nous avons adoptée dans le cadre de nos travaux repose essentiellement sur **le parallélisme des données** sur la partie convolutive. Pour ce faire nous avons mis en oeuvre cette stratégie ainsi que suit :

Approche : Il s'agit ici de paralléliser le modèle au niveau de la partie convolutive ou extraction des caractéristiques selon le parallélisme des données tout en laissant la couche entièrement connectée sous sa forme séquentielle comme l'illustre la figure 3.1. En

fait, la partie extraction des caractéristiques repose sur 2 opérations principales à savoir la convolution et le pooling. Mais la **convolution** est l'opération qui a la complexité ($O(L * H * F^2 * N_{can})$) la plus élevée dans tout le CNN et prends donc plus de temps lors de l'entrainement du modèle. Par ailleurs, lors de la phase d'entrainement nous avons utilisé la descente de gradient version mini-batch pour l'optimisation du modèle. Pour davantage étayer notre idée, nous avons procédé ainsi que suit :

- Décomposer le jeu de données en plusieurs lots ;
- Lors de la phase avant (forward) dans la partie convolutive, pour chaque lot, les threads travaillent sur chaque image (portions distinctes de l'image) pour effectuer la convolution, le pooling afin d'obtenir les vecteurs de caractéristiques correspondants aux différentes images ;
- Effectuer le forward au niveau de la partie entièrement connectée de façon séquentielle ;
- La rétropropagation s'effectue de façon séquentielle au niveau de la couche entièrement connectée et de façon parallèle sur la partie convolutive.

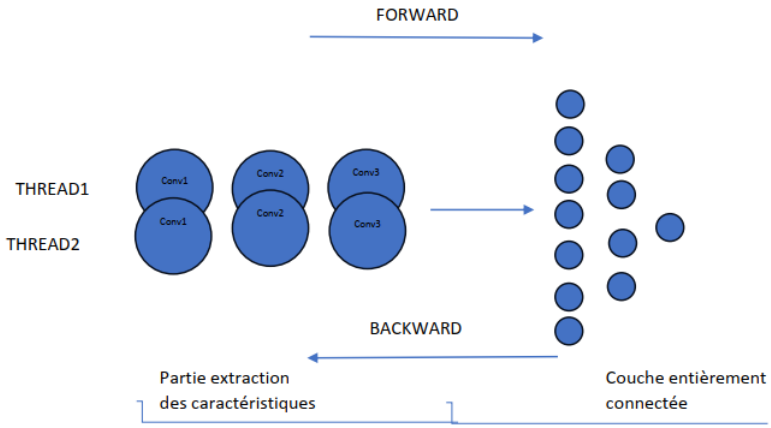

FIGURE 3.1 – Illustration de la stratégie avec 2 threads.

Dans la figure 3.1, chaque thread reçoit l'ensemble des fonctions liées à la partie convolutive afin d'effectuer l'extraction des caractéristiques sur les lots d'images. Cela se matérialise par la production des vecteurs de caractéristiques de chacun des images des lots. Une fois ces vecteurs obtenus, la classification peut commencer de façon séquentielle sur la couche entièrement connectée.

3.3 Algorithme Parallèle

Pour cette stratégie de parallélisation, la principale modification est effectuée au niveau de l'opération de convolution au niveau de la phase de propagation avant et arrière. Dans notre stratégie nous maintenons l'agorithme d'entrainement mais avec **la convolution parallèle** dans le Forward et le Backforward comme suit :

Algorithm 4 : **Training Parallele**

Input (X,Y) : ensemble d'images x et leur label y ; λ : le learning rate ;
M : le batch size ; epoch : le nombre d'époques maximal
Output Θ : Ensemble de paramètres du modèle
Start

1: initializeParameters(Θ) *// on initialise les paramètres Θ du modèle*
2: (xtrain, ytrain) (xval, yval) $\leftarrow$ splitData(X, Y) *// on réserve une partie du jeu de données pour la validation*
3: B $\leftarrow$ getBatch(M,xtrain,ytrain) *//on crée des lots de données de taille M*
4: e $\leftarrow$ 1
5: stop $\leftarrow$ 0
6: valLoss $\leftarrow \infty$
7: **while** $e \leq$ epoch **and** stop < 4 **do**
8: **for** $\mathsf{b_i} \in \mathsf{B}$ **do**
9: loss $\leftarrow$ 0
10: zeroInitialize($\mathsf{d\Theta}$) *// pour chaque lots de données, on initialise les dérivées partielles à zéro*
11: **for** $(\mathsf{x}, \mathsf{y}) \in \mathsf{b_i}$ **do**
12: $\mathsf{y_{pred}} \leftarrow \mathsf{Forward_{parallele}}(\mathsf{x}, \Theta)$ *// pour chaque phrase x dans un lot de donnée, l'on prédit sa classe*
13: $\mathsf{loss} \leftarrow \mathsf{loss} + \mathsf{lossEntropy}(\mathsf{y}, \mathsf{y_{pred}})$ *// on accumule l'erreur de prédiction*
14: $\mathsf{d\Theta} \leftarrow \mathsf{d\Theta} + \mathsf{Backforward_{parallele}}(\Theta, \mathsf{x}, \mathsf{y}, \mathsf{y_{pred}})$ *//on calcul et on accumule les dérivées partielles*
15: **end for**
16: update($\Theta, \mathsf{d\Theta}, \lambda, \mathsf{M}$) *//on mets à jour les paramètres*
17: **end for**
18: **if** evalModel(Θ, xval, yval) $>$ valLoss **then**
19: stop $\leftarrow$ stop + 1 *// si les performances du modèle n'ont pas augmenté, on incrémente la variable stop*
20: **else**
21: $\mathsf{valLoss} \leftarrow \mathsf{evalModel}(\Theta, \mathsf{xval}, \mathsf{yval})$
22: stop $\leftarrow$ 0 *// si non, la variable stop est réinitialisé à zéro*
23: **end if**
24: e $\leftarrow$ e + 1
25: **end while**
26: **Return** Θ

End

Cet algorithme, nous présente l'exécution parallèle de l'entrainement de notre modèle de CNN parallèle. Il se démarque grace aux lignes 12 et 14.

Ensuite, la modification se poursuit dans l'opération de convolution pour le forward comme suit :

Algorithm 5 : Forward Parallele

Input x : une image du jeu de données ;
Θ : les paramètres de la partie entièrement connectée ;
γ : Les paramètres de la partie convolutives
Output y_{pred} : Valeur prédite par le modèle
Start

1: i ← 1
2: **while** i <= nbrecouche **do**
3: x ← $\mathsf{Convolution}_{\mathsf{Parallele}}$(x,$\gamma$)
4: x ← $\mathsf{Fonction}_{\mathsf{activation}}$(x) *// très souvent Relue ou TanH*
5: x ← $\mathsf{Pooling}_{\mathsf{Parallele}}$(x)
6: i ← i + 1
7: **end while**
8: V ← Vectorisation(x)
9: Z ← Θ .V + b_{FC}
10: y_{pred} ← f(Z) *// f est une fonction d'activation qui peut être selon le cas sigmoïde ou softmax ...*
11: **Return** y_{pred}
End

Algorithm 6 : Backforward Parallele

Input x : une image du jeu de donnée ; y_{pred} : Valeur prédite par le modèle ;
y_{train} :label de l' image ; Θ : les paramètres de la partie entièrement connectée ;
γ : Les paramètres de la partie convolutive
Output dΘ : Ensemble des dérivées partielles du modèle
Start

1: n ← LENGTH(x)
2: ZeroInitialiZe(dΘ, dγ) *// On initialise les gradients à zéro*
3: $dW \leftarrow \mathsf{Vect}_{\mathsf{image}}^{\mathsf{Transpos}}.(y_{pred} - y_{train})$ *// dW matrice de gradients des poids de la couche entièrement connectée*
4: $db_{FC} \leftarrow \sum(y_{pred} - y_{train})$ *// db_{FC} gradients des biais de la couche entièrement connectée*
5: $dx \leftarrow \mathsf{Convolution}_{\mathsf{P}}\mathsf{arallele}(\mathsf{padded}(dO), \mathsf{filtre}_{\mathsf{roter-de-180}})$ *// dO gradients propagés provenant de la couche précédente*
6: $dF \leftarrow \mathsf{Convolution}_{\mathsf{P}}\mathsf{arallele}(x, dx)$ *// dF gradients des filtres et dx est la dérivée partielle de l'image*
7: $db_C \leftarrow \sum(dO)$ *// db_C gradient des biais de la partie convolutive*
8: **Return** $\{dW, dF, db_{FC}, db_C\}$
End

Algorithm 7 : Convolution Parallele

Input (x) : une image du jeu de données avec une taille L x H ;
F : un filtre de taille L_F x H_F ;
can : le nombre de canal de l'image ; T : le nombre de thread
Output img : une carte de caractéristique (image convoluée)
Start

1: **for** t = 0 ; t < T ; t++ **do**
2: **for** i = t ; i < L ; i += T **do**
3: **for** j=0 ; j < H ; j++ **do**
4: **for** k=0 ; k < L_F ; k++ **do**
5: **for** m=0 ; m < H_F ; m++ **do**
6: e ← i*pas+k
7: z ← j*pas+t
8: **for** c=0 ; c < can ; c++ **do**
img.p[i][j].pixel[c] += x.p[e][z].pixel[c] * F.p[k][m].pixel[c]
9: **end for**
10: **end for**
11: **end for**
12: **end for**
13: **end for**
14: **end for**
15: Return **img**
End

Cet algorithme représente la convolution parallèle effectuée par notre modèle. La ligne 1 nous permet de manipuler les threads ; la ligne 2 quant à elle nous permet d'attribuer chaque thread à des lignes ou portions spécifiques de l'image d'entrée. Les lignes 3 à 8 nous permettent d'effectuer la convolution simultanément entre notre image et le filtre associé.

Chapitre 4

Expérimentations et Résultats

Dans ce chapitre, nous présenterons tout d'abord notre cadre expérimental. Nous présenterons ensuite les résultats expérimentaux obtenus pour l'exécution séquentielle en se servant des métriques de performances telles que l'accuracy et le temps d'exécution. Pour finir nous présenterons les résultats de l'exécution parallèle en utilisant des mesures de performances telles que le *speedup* et le temps d'exécution concernant la première approche de parallélisation.

4.1 Cadre expérimental

Pour ces résultats préliminaires, les expérimentations ont été faites sur une machine multi-cœur ayant les caractéristiques suivantes : **8 cœurs à 1,80 GHz**, 16 Go de Ram. Les implémentations des algorithmes d'entraînement ont été effectuées en langage C++ et en utilisant la bibliothèque *posix thread* pour les implémentations parallèles.

Pour les expérimentations, nous avons utilisé un jeu de données constitué d'images de lame sanguine préalablement colorés à l'aide d'une solution chimique (généralement Giemsa) obtenu à partir d'un microscope avec lequel les cellules peuvent etre facilement observables. Le jeu de donnée contient 117 images dont 62 infectées et 55 non infectées. Afin de pouvoir les utiliser, nous avons changées le format des images en .ppm et nous les avons redimensionnées à 750x750 pixels. Après l'avoir pré-traité, le jeu de données a ensuite été divisés en deux, à savoir 80% pour l'entraînement, et 20% pour le test.

4.2 Protocole expérimentale & Résultats

Pour mener à bien nos expérimentations, nous avons défini l'architecture suivante pour notre réseau de neurone convolutif :

- Le nombre d'époque maximale : 5
- Le learning rate λ : 0.005
- Le batch size M : 5
- Le nombre de couche convolutive est de 3
- Le nombre total de neurones $\mathsf{N_h}$ dans la couche cachée : 41.

Afin d'évaluer la performance de notre modèle de classification, plusieurs métriques peuvent etre utilisées. Il s'agit de l'accuracy, la précision, le rappel etc. Pour notre cas, nous nous sommes attardé sur l'**accuracy** qui est défini grace aux valeurs suivantes :
VP (Vrai Positif) : Il s'agit du nombre d'éléments catégorisés dans une classe à laquelle ils appartiennent effectivement.
FP (Faux Positif) : C'est le nombre d'éléments catégorisés dans une classe à laquelle ils n'appartiennent pas.
FN (Faux Négatif) : C'est le nombre d'éléments appartenant à une classe mais ayant été assignés à une autre classe.
VN (Vrai Négatif) : C'est le nombre d'éléments ayant été correctement reconnus comme n'appartenant pas à la classe.
L'accuracy permet de décrire la performance du modèle sur les individus positifs et négatifs de façon symétrique. Elle mesure le taux de prédictions correctes sur l'ensemble des individus. Il se défini ainsi qu'il suit :

$$accuracy = \frac{VP + VN}{VP + FP + FN + VN} \tag{4.1}$$

Après exécution de la version séquentielle, nous avons obtenu une accuracy de 50% et 5155.88 secondes (1h25min55s) comme temps d'exécution.

4.2.1 Programmation Parallèle

Le paradigme de programmation parallèle désigne l'ensemble de méthodes permettant de prendre avantages des architectures multi-cœurs pour écrire des programmes avec des instructions s'exécutant simultanément, afin de réduire le temps d'exécution. La possibilité d'écrire un programme parallèle dépend fortement de l'architecture matérielle de la plate-forme d'exécution [14], et du modèle de programmation parallèle que l'on désire implémenter.

4.2.2 Mesure de performances d'un calcul parallèle

Soit p le nombre de ressources d'une machine parallèle. Une ressource correspond à un cœur de calculs dans un processeur multi-coeurs ou à un processeur dans une machine multiprocesseurs, ou même à un nœud (PC) dans un cluster. Les performances dues à la parallélisation d'un algorithme seront mesurées à travers :

- **Le temps d'exécution T(p)** : C'est le temps d'exécution mit par un programme parallèle en utilisant p ressources.
- **Le *Speedup* S(p)** : C'est le rapport entre le temps d'exécution avec une ressource, sur le temps d'exécution sur p ressources.

$$\mathsf{S(p)} = \frac{\mathsf{T(1)}}{\mathsf{T(p)}} \tag{4.2}$$

Lorsqu'on effectue une bonne parallélisation, on devrait avoir une valeur de S(p) comprise entre 1 et p, en espérant qu'elle soit le plus proche de p.

En utilisant les huit coeurs disponibles, l'on obtient un *speedup* de 2,085 et un *temps d'exécution* de 2472.11 secondes (41min12s) lors de l'entrainement notre stratégie.

La figure suivante présente le temps d'exécution et le *speedup* en fonction du nombre de *threads* (cœurs) utilisés.

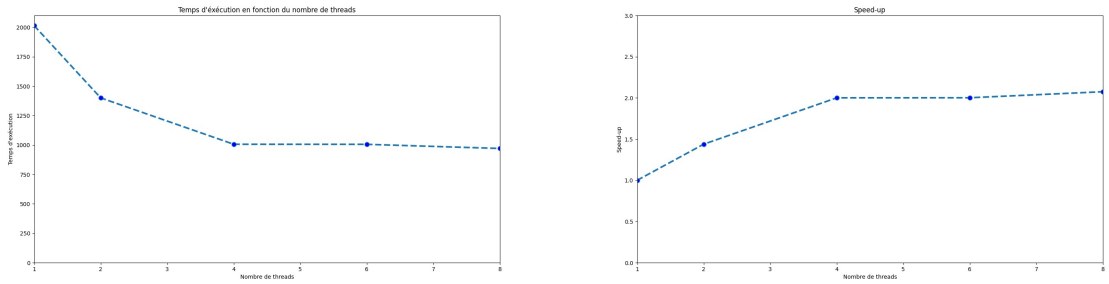

FIGURE 4.1 – *speedup* et temps d'exécution pour l'entraînement d'un CNN en fonction du nombre de thread (nombre d'unité de traitements) notre dataset

4.3 Commentaire & Discussion

Au regard des travaux existants et des différentes expérimentations menées au cours de ce travail nous avons pu voir que plusieurs facteurs tels que l'architecture du CNN, taille du jeu de données, architecture matérielle, stratégie de parallélisation, les algorithmes utilisés pour implémenter le CNN, le nombre de threads utilisés, le choix des fonctions à paralléliser peuvent influencer la parallélisation d'un modèle de CNN. Dans [10] l'auteur

utilise une stratégie basée sur des GPUs et obtient avec 8 unités de traitements un speedup de 6.16 en utilisant le parallélisme de données sur la couche convolutive et le parallélisme de modèle sur la couche entièremment connecté alors que nous utilisons juste le parallélisme de données sur la couche convolutive car le parallélisme de modèle dispose de certains inconvénients comme l'apparition d'un temps *bulle* pendant lequel certains appareils ou unité de traitement ne sont pas engagés dans le calcul, ce qui entraînera un gaspillage de ressources de calcul [19] et l'obsolescence des poids, c'est-à-dire que les gradients sont calculés à un moment donné avec des poids obsolètes, ce qui conduit à instabilité de l'entrainement et une perte de précision [3]. La différence d'équipement matériel (CPUs au détriment des GPUs) justifierait en parti l'écart de nos résultats.

Chapitre 5

Conclusion et perspectives

Nous présenterons dans ce chapitre un résumé des objectifs de nos travaux, les résultats préliminaires que nous avons obtenus, ainsi que des directives futures.

5.1 Conclusion

L'objectif de ce mémoire était d'implémenter de façon parallèle l'algorithme d'entraînement d'un CNN pour la détection du paludisme. Pour le faire, nous avons tout d'abord présenté de façon générale le principe et fonctionnement ainsi que l'architecture d'un CNN. Nous avons ensuite procédé à une exécution séquentielle en utilisant un jeu de donnée d'images de frottis sanguins de patients infectés et non infectés par la maladie. Pour ce jeu de données, l'on a obtenu un modèle avec 50% d'accuracy. Nous avons ensuite procédé à une implémentation et exécution parallèle de l'algorithme d'entraînement des CNN en utilisant le meme jeu de données en employant le parallélisme de donnée sur la partie d'extraction des caractéristiques. Avec cette version nous avons pu reduire le temps d'entrainement du modèle avec un speedup de 2,085 tout en maintenant l'accuracy.

5.2 Perspectives

Au terme de ce travail, plusieurs perspectives émergent pour améliorer notre recherche. Tout d'abord, il est essentiel d'étendre notre approche de parallélisation de CNN à des ensembles de données plus vastes et complexes afin de mieux évaluer son efficacité et sa généralisation. Ensuite, il serait pertinent d'explorer d'autres cadres pour la parallélisation telles que la parallélisation sur environnement distribué et l'utilisation de matériels spécialisés pour améliorer les performances des CNN. Par ailleurs, l'utilisation d'autres

stratégies comme le parallélisme de données sur l'ensemble du modèle ou le parallélisme de données à deux niveaux (sur l'ensemble du CNN et dans sa partie convolutive) permettrait d'avantage d'enrichir et approfondir nos expérimentations. Enfin, nous souhaitons envisager des applications plus spécifiques dans d'autres domaines de la recherche médicale.

Bibliographie

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow : Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv :1603.04467*, 2016.

[2] Niccolò Cacciotti. Computer vision : the ultimate guide on the 4 main tasks, 2022. [En ligne ; Page disponible le 22-juillet-2022].

[3] Hsiang-Yun Cheng Chi-Chung Chen, Chia-Lin Yang. Efficient and robust parallel dnn training through model parallelism on multi-gpu platform. 2019.

[4] Selima Curci, Decebal Constantin Mocanu, and Mykola Pechenizkiyi. Truly sparse neural networks at scale. *arXiv preprint arXiv :2102.01732*, 2021.

[5] DataScientest. Convolutional neural network : Tout ce qu'il y a à savoir, 2023.

[6] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Marc'aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, et al. Large scale distributed deep networks. *Advances in neural information processing systems*, 25, 2012.

[7] Hang Yu Zhou Zhou Kamolrat Silamut Jian Yu Richard J Maude Stefan Jaeger* Sameer Antani Feng Yang, Mahdieh Poostchi. Deep learning for smartphone-based malaria parasite detection in thick blood smears. 2019.

[8] Luhui Hu. Distributed parallel training : Data parallelism and model parallelism, 2022.

[9] Belajar Pembelajaran Mesin Indonesia. Student notes : Convolutional neural networks (cnn) introduction, 2018. [En ligne ; Page disponible le 07-mars-2018].

[10] Alex Krizhevsky. One weird trick for parallelizing convolutional neural networks. 2014.

[11] World Health Organization. World malaria report 2021, 2021.

[12] Thomas Paine, Hailin Jin, Jianchao Yang, Zhe Lin, and Thomas Huang. Gpu asynchronous stochastic gradient descent to speed up neural network training. *arXiv preprint arXiv :1312.6186*, 2013.

[13] Georgios K Pitsilis, Heri Ramampiaro, and Helge Langseth. Effective hate-speech detection in twitter data using recurrent neural networks. *Applied Intelligence*, 48 :4730–4742, 2018.

[14] T Rauber and G Rünger. Parallel programming : For multicore and cluster systems.[sl] : Springer science & business media, 2013. *Citado na*, page 30, 2013.

[15] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv :1609.04747*, 2016.

[16] Dipanjan (DJ) Sarkar. Detecting malaria with deep learning, 2019.

[17] Saily Shah. Convolutional neural network : An overview, 2022.

[18] Diksha Sharma and Neeraj Kumar. A review on machine learning algorithms, tasks and applications. *International Journal of Advanced Research in Computer Engineering & Technology (IJARCET)*, 6(10) :2278–1323, 2017.

[19] Siqi Mai Shenggui Li. Paradigms of parallelism, 2023.

[20] Ankit Agrawal Alok Choudhary Sunwoo Lee, Dipendra Jha and Wei keng Liao. Parallel deep convolutional neural network training by exploiting the overlapping of computation and communication. In *2017 IEEE 24th International Conference on High Performance Computing*, pages 183–192. IEEE, 2017.

[21] Yugesh Verma. A guide to parallel and distributed deep learning for beginners, 2021.

[22] Yuhang Dong W. David Pan and Dongsheng Wu. Classification of malaria-infected cells using deep convolutional neural networks. *Machine Learning - Advanced Techniques and Emerging Applications*, pages 159–174, 2018.

[23] Wikipédia. Cross industry standard process for data mining — wikipédia, l'encyclopédie libre, 2022. [En ligne ; Page disponible le 30-juin-2022].