Projet OCR

Rapport de Soutenance n°1

Grégoire Le Bihan, Dardan Bytyqi
Léo Menaldo, Nathan Fontaine

Table des matières

1 Prélude

Aujourd'hui, en ouvrant un journal ou un magazine, on trouve toujours une ou deux pages remplies de casses-tête que ce soit des mots croisés, des mot fléchés... Mais on y trouve aussi des grilles de sudoku, un jeu très simple par abord, mais qui est pourtant extrêmement populaire, en effet, pas besoin d'avoir la bosse des maths pour terminer une grille à 100%, il suffit simplement d'avoir de la logique.
Dans les années 2000, les sudokus sont apparus dans les journaux, et ont amené le jeu sous un nouveau jour. Depuis, des informaticiens se sont penchés sur le sujet et ont essayé de résoudre des sudokus à l'aide d'un programme informatique, ils ont donc créé plusieurs algorithmes, certains se rapprochant de la façon dont un joueur résout la grille, d'autres étant un peu plus abstraits et très calculatoires.
Dans le cadre du projet de troisième semestre à l'EPITA, on nous a confié la mission de devoir être capable de résoudre une grille de sudoku présente sur une image, quelle que soit l'image.

2 Assignation des tâches

Ci-dessous, les tâches assignées à chacun des membres du groupe :

2.1 Dardan Bytyqi

Le pré-traitement de l'image doit être fait, l'image d'entrée doit devenir une image monochrome.

2.2 Nathan Fontaine

Une fois l'image monochrome obtenue, il faudra découper chacune des cases de l'image.

2.3 Léo Menaldo

Un réseau de neurones devra être implémenté, pour le moment, elle sera capable d'apprendre la fonction XOR.

2.4 Grégoire Le Bihan

Un algorithme de résolution de sudoku doit être implémenté, la rotation d'image doit elle aussi être implémentée.

3 Avancement du projet

3.1 Le pré-traitement de l'image

3.1.1 Flou et binarisation

Le pré-traitement de l'image est le premier processus dont le projet a besoin, son objectif est de rendre l'image plus simple à traiter pour la suite. Les éléments qui nous posent problème sur les images et que l'on souhaite éliminer sont le bruit et les gradients, on cherchera alors des méthodes pour obtenir des images binarisées (images composées uniquement de noir ou de blanc mais pas de gris) sans bruit pour rendre la lecture plus simple.
Il existe alors de nombreuses méthodes pour binariser des images, et après de nombreuses recherches et implémentations, il a été décidé d'utiliser le filtre de Canny. Il faut alors premièrement mettre en place un flou de Gauss. Il s'agit d'une méthode qui permet de flouter une image en appliquant une convolution a cette image avec une distribution de Gauss. Cette technique permet de garder une image relativement détaillée, c'est a dire que l'on ne perd pas trop d'informations, tout en réduisant grandement la quantité de bruit de l'image. On voit bien ici

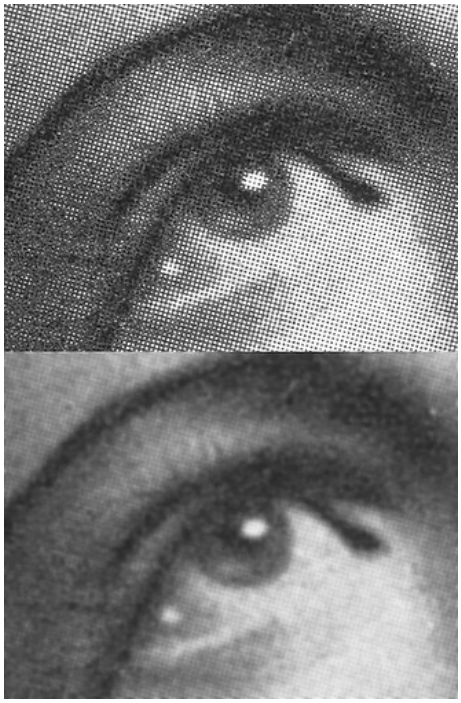

FIGURE 1 – Exemple de réduction de grain grâce au flou de Gauss

que le grain de l'image est beaucoup moins visible, et par conséquent, elle sera plus simple à traiter. Après avoir réduit le bruit de l'image, nous chercherons a faire une binarisation de l'image. Pour cela on utilisera une méthode à double seuil, cette technique consiste a avoir deux

seuils, un seuil bas et un seuil haut, tous les pixels dont la valeur en grayscale est inférieure au seuil bas seront convertis en pixels noirs, tous les pixels dont la valeur en grayscale est supérieure au seuil haut seront convertis en pixels blancs, et pour tous les pixels entre les deux seuils, on cherchera à savoir si ils ont des pixels au dessus du seuil haut comme voisins proches, si c'est le cas, alors on les considère comme des pixels blancs, sinon, comme des pixels noirs.
On ne veut cependant pas avoir de seuils fixes sur les différentes images, on utilisera alors la valeur médiane de l'image (la médiane des valeurs grayscales de l'image) pour en déduire les deux seuils.

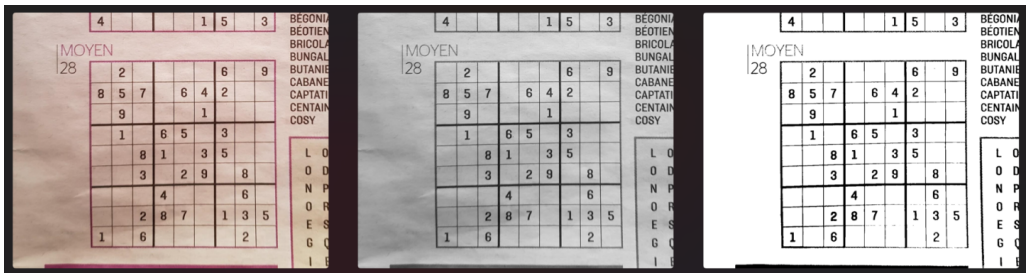

FIGURE 2 – Exemple du traitement jusqu'à la binarisation

3.1.2 Rotation de l'image

La rotation d'une image est une opération courante réalisée à l'aide de la bibliothèque SDL. Cette bibliothèque offre des fonctionnalités avancées de traitement d'image. Au cours de cette année, nous avons pu observer l'utilité de cette bibliothèque lors de nos travaux pratiques. En explorant la documentation de SDL, j'ai découvert une fonction particulièrement utile : SDL_RenderCopyEx. Après lecture de la documentation, j'ai constaté que cette fonction permettait de copier une image tout en l'inversant ou en la faisant pivoter autour de son centre. Le choix d'utiliser cette fonction s'explique par sa simplicité d'utilisation pour la rotation d'image, ce qui facilite grandement le processus. Cependant, il est important de noter que l'utilisation de SDL nécessite la création d'une fenêtre, ce qui peut avoir un impact sur les ressources système, car la bibliothèque maintient cette fenêtre ouverte pendant l'exécution du programme. Cela implique probablement une utilisation de la mémoire qui aurait pu être allégée si nous avions opté pour une approche basée sur une simple matrice de rotation. Cependant, il est crucial de souligner que la création d'un lien entre la matrice de rotation et le pivot, tout en calculant la distance entre chaque pixel et le centre de l'image, est une tâche complexe qui exige une expertise avancée en traitement d'images. À mon niveau actuel, cette complexité rendrait difficile l'optimisation de l'algo-

rithme. En fin de compte, le choix de la fonction SDL_RenderCopyEx repose sur la balance entre la facilité d'utilisation et les ressources système requises. Bien que d'autres approches puissent être plus efficaces sur le plan des ressources, elles peuvent être plus complexes à mettre en œuvre. Pour des projets plus avancés, il pourrait être intéressant d'explorer des solutions plus optimisées, mais cela nécessiterait une expertise supplémentaire en traitement d'image et en optimisation d'algorithme.

3.2 Le traitement de l'image

3.2.1 Détection des lignes

La question de la détection des lignes sur une image est un problème difficile. En effet il faut tout d'abord utiliser une image monochrome composée essentiellement de points, de plus ces images seront parfois incomplètes, en effet, selon la qualité de l'image d'origine, les images seront composées de plus ou moins de points et de bruits, pour faire face à ce problème, nous avons utilisé une technique appelée "transformée de Hough". Il existe deux procédés différents :

- Une méthode avec un accumulateur contenant deux paramètres (m, c). Ce qui équivaut à avoir un point de coordonnées (x, y) quelconques, qui peut être contenu dans une droite de fonction affine $y = mx + c$ avec m un multiplicateur et c une constante.
- Une autre méthode qui utilise elle aussi un accumulateur, mais cette fois de paramètres (θ, ρ). Ce qui équivaut à avoir un point de coordonnées (x, y) quelconques, qui peut être contenu dans une droite sur le plan paramétrique trigonométrique avec $x \times sin(\theta) - y \times cos(\theta) + \rho = 0$ avec ρ l'ordonnée à l'origine sur le plan paramétrique et θ l'angle par rapport à l'axe horizontal.

Méthode avec un accumulateur (m, c) :
Mise en place : L'accumulateur (m, c) stocke le nombre de points qui satisfont l'équation de chaque ligne. Les valeurs de m et de c sont choisies de manière à couvrir toutes les lignes potentielles dans l'espace bidimensionnel.
Avantages : Cette méthode est relativement simple à comprendre et à mettre en œuvre. Elle peut être efficace pour détecter des droites dans des espaces où les lignes sont bien représentées par des équations linéaires. De plus, les paramètres m et c peuvent être interprétés directement comme les caractéristiques de la droite détectée.
Problèmes : Cette méthode peut être limitée dans sa capacité à détecter des lignes qui ne peuvent pas être facilement représentées par une équa-

tion linéaire. De plus, elle peut rencontrer des problèmes de précision lorsque les lignes ne sont pas parfaitement droites ou lorsque les données sont sujettes à du bruit ou des imprécisions. Qui plus est, lorsque l'image est grande, des erreurs de "Segmentation fault" peuvent survenir de manière inexplicable et incongrue, ce qui m'a poussé à me tourner vers la seconde méthode. De plus cette méthode consomme beaucoup plus de mémoire étant donné qu'elle génère énormément de fois plus de droite suite aux différents m et c possibles.

Méthode avec un accumulateur (θ, ρ) :
Mise en place : L'accumulateur (θ, ρ) stocke le nombre de points qui satisfont l'équation de chaque ligne. Les valeurs de θ et de ρ sont choisies pour couvrir toutes les lignes potentielles dans l'espace bi-dimensionnel.
Avantages : Cette méthode est robuste pour détecter des lignes dans des espaces où les droites peuvent avoir différentes orientations. Elle peut mieux gérer les cas où les lignes ne sont pas parfaitement droites ou lorsque les données contiennent du bruit.
Problèmes : La méthode (θ, ρ) peut être plus complexe à mettre en œuvre que la méthode (m, c), car elle implique des opérations trigonométriques. De plus, l'interprétation des paramètres θ et ρ peut être moins directe que celle de m et c dans certaines applications.

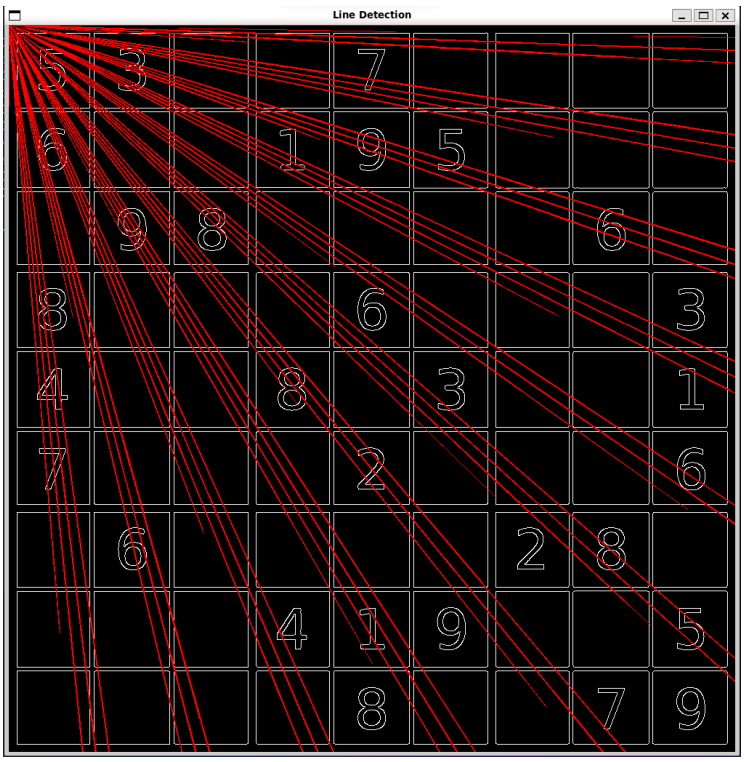

FIGURE 3 – Premiers test de détection de ligne ratés

3.2.2 Découpage de l'image

Pour arriver au découpage de l'image, il va tout d'abord falloir supprimer toutes les lignes inutiles ce qui revient à reconnaître la grille du sudoku dans l'image.
Pour cela, on utilise les paramètres représentant les droites (donc (θ, ρ)) et on va faire un parcours sur l'accumulateur et supprimer toutes les lignes qui se confondent pour ainsi obtenir des tracés uniques.
Ensuite, pour savoir où se trouve la grille du sudoku, on regarde s'il y a des droites qui possèdent une intervalle régulière entre elles et normalement on devrait garder seulement les lignes de la grille du sudoku. Ici, étant donné que la détection des lignes est approximative, il faut avoir un seuil de valeur approprié à chaque image pour pouvoir reconnaître toutes les lignes de manière optimales, cette valeur de seuil ne doit être ni trop faible (car sinon, il est plus probable de manquer des lignes de la grille du sudoku) ni trop élevée (car sinon, il restera des lignes qui ne sont pas celles de la grille du sudoku).
Dès lors que cette étape à bien été réalisée, il faut ensuite calculer les zones d'intersections entre les lignes représentant les cases du sudoku, ensuite grâce à SDL, on enregistre une sous-image à partir des coordonnées qu'on a calculées puis on l'enregistre selon ces coordonnées dans la matrice représentant le sudoku afin de pouvoir bien les traiter dans le passage à l'IA.
Mais, il faut garder en mémoire que toutes ces étapes requièrent des ajustements spécifiques en fonction de l'image, en effet si l'image possède du bruit, il va falloir faire une étape supplémentaire pour traiter les cases vides qui peuvent parfois laisser passer des cases vides. Un autre problème apparaît lorsque l'image d'origine n'est pas de face par rapport au sudoku, alors il faudra cette fois implémenter un algorithme de redressement d'image afin d'avoir des images découpées les plus carrées possible pour les transmettre au réseau neuronal.
En conclusion, le processus de découpage de l'image pour la reconnaissance de la grille du sudoku est un processus complexe qui nécessite une série d'étapes délicates. La détection des lignes à l'aide des paramètres (θ, ρ) suivi du nettoyage des lignes inutiles est crucial pour identifier la grille du sudoku dans l'image. La détermination d'un seuil approprié pour la détection des lignes et la gestion des images bruitées sont des aspects essentiels pour obtenir des résultats fiables. De plus, la détection précise des intersections entre les lignes et la prise en compte des distorsions dans l'orientation de l'image originale ajoutent une complexité supplémentaire. Ainsi, la mise en œuvre réussie de cet algorithme néces-

site une compréhension approfondie des caractéristiques spécifiques de l'image à traiter et la capacité d'ajuster les paramètres en conséquence pour obtenir des résultats optimaux.

3.3 Le réseau neuronal

Pour reconnaître des caractères présents dans l'image, il est primordial d'utiliser un réseau neuronal, le principe est simple, on doit lui faire comprendre que pour certaines entrées, il doit nous donner certaines sorties.
Pour cette soutenance, il faut être capable de faire apprendre la fonction XOR à notre réseau.

3.3.1 Recherche et réflexion

La recherche a été un travail complexe à réaliser, en effet, il existe énormément d'articles, de vidéos et de recherches à ce sujet. La première étape a donc été de lire le **livre** sur les réseaux neuronaux et l'apprentissage supervisé de Michael Nielsen, un cours très bien détaillé qui explique parfaitement l'origine et la conception des réseaux neuronaux. Il a fallu lire le livre plusieurs fois pour bien assimiler le concept et être capable de le reproduire par la suite. Afin de débloquer encore plus de clés de compréhension, des vidéos publiées par 3Blue1Brown sont disponibles sur **YouTube** et résument parfaitement le contenu du livre de Michael Nielsen, de manière visuelle et présentée sous un autre angle.
Pour ce qui est de la réflexion, il a fallu se demander par quel moyen allait être organisé le code, comment allait-on représenter un réseau neuronal, ses poids, ses biais. Utilise-t-on des structures de données ? Des tableaux ? Des matrices ?

3.3.2 Comment se servir du réseau neuronal ?

Pour le bon fonctionnement du réseau neuronal. Il a été préférable d'utiliser des matrices. Le réseau neuronal peut être représenté en utilisant des matrices pour tout ce qu'il contient, par exemple, les couches de neurones peuvent exister sous forme de matrices colonnes, et étant donné que les calculs effectués dans un perceptron sont de la forme :

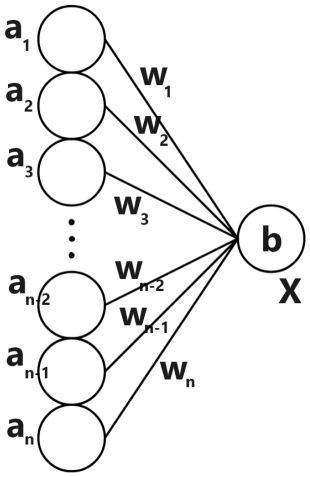

FIGURE 4 – Perceptron X

$$X = b + \sum_{i=1}^{n} a_i \times w_i$$

On a donc : $X = \begin{bmatrix} w_1 & w_2 & \dots & w_{n-1} & w_n \end{bmatrix} \times \begin{bmatrix} a_1 \\ a_2 \\ \dots \\ a_{n-1} \\ a_n \end{bmatrix} + b$

Les opérations sont plutôt adaptées à la situation.
Pour le réseau neuronal, il est préférable de faire une structure de données qui contiendra l'intégralité du réseau neuronal, autrement dit, combien de couches possède le réseau, combien de noeuds possède chaque couche, la valeur de tous les poids du réseau et les biais des couches différentes de la couche d'entrée.

3.3.3 Méthode d'apprentissage

Après tout ce qui a été acquis, il est temps de voir quelle méthode a été utilisée pour que le réseau neuronal apprenne. La façon dont le réseau apprend est la suivante :

- Le réseau prend les valeurs données et calcule le résultat.
- Selon le résultat, il faudra changer très peu ou beaucoup les biais et les poids du réseau.
- On calcule donc le gradient en appliquant la dérivée de la fonction sigmoïde à la sortie obtenue.
- À partir du gradient et des couches existantes, on calcule des valeurs infinitésimales qui nous serviront à modifier légèrement les poids et biais du réseau.

— Enfin, on modifie les poids et biais du réseau.

Ce processus nous permet d'avoir des résultats toujours différents, mais aussi, toujours bons.

3.3.4 La fonction XOR

Pour la fonction XOR, il a fallu se poser plusieurs questions essentielles. On connaît la fonction XOR, on sait que pour certaines entrées, on obtient certaines sorties, mais pour autant, il faut quand même se pencher un peu plus à ce sujet. Après quelques recherches, on découvre que la fonction XOR est un problème "Non linéairement séparable".

Mais qu'est ce qu'un problème "Linéairement séparable" ? On peut prendre l'exemple de la fonction ET et de la fonction OU. Ces deux fonctions

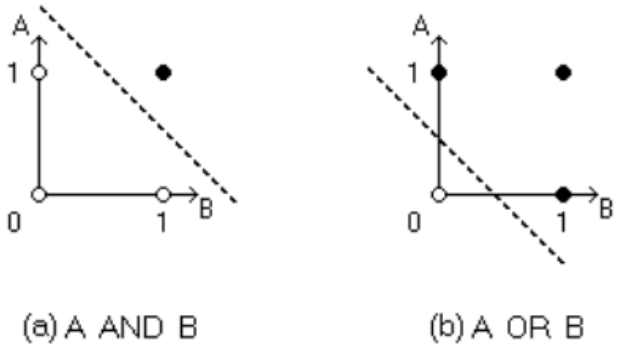

FIGURE 5 – Fonction ET et OU, des fonctions linéairement séparables

sont linéairement séparables car il est possible de séparer les résultats justes et les résultats faux. C'est là qu'on rencontre un problème, il est

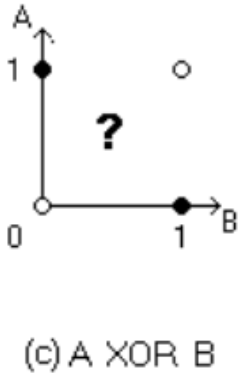

FIGURE 6 – Fonction XOR,une fonction qui n'est pas linéairement séparable

impossible d'apprendre la fonction XOR à notre réseau, il lui manque un élément... On pourrait passer par l'équivalent de la fonction, en disant que $A \oplus B = (A \wedge \neg B) \vee (\neg A \wedge B)$ et apprendre cet équivalent au réseau,

mais c'est en passant par cette option qu'on remarque que le réseau peut utiliser des fonctions linéairement séparables pour apprendre une fonction non linéairement séparable. C'est ainsi qu'on en déduit le fait que notre réseau devra posséder une couche cachée, une couche supplémentaire qui servira d'intermédiaire entre la couche d'entrée et la couche de sortie.

3.3.5 Conclusion

Pour faire fonctionner notre réseau de neurones et faire en sorte qu'il apprenne la fonction XOR, la meilleure option était de créer un réseau neuronal à 3 couches.

— Une couche d'entrée : Contenant 2 neurones
— Une couche cachée : Contenant 4 neurones
— Une couche de sortie : Contenant 1 neurone

Ce réseau, auquel on initialise toutes les valeurs à des valeurs aléatoires, va être entraîné un certain nombre de fois, ce qui va permettre de régler les valeurs choisies aléatoirement, plus ce nombre est grand plus le réseau donnera des résultats précis. Il apprendra plus ou moins vite selon son taux d'apprentissage qui doit être choisi minutieusement. Dans le contexte du XOR, celui-ci sera de 0.1. Une fois l'entraînement terminé, on peut donner n'importe quelle série d'entrées, et on obtiendra la sortie escompté.

3.4 La résolution du sudoku

Pour la résolution du sudoku, plusieurs pistes on été approfondies. Chacune ayant ses avantages. Certaines pistes seront simplement évoquées dans ce document car elles n'ont pas été suffisamment approfondies : La méthode brute-force (car peu optimisée), les algorithmes génétiques, de recherche heuristiques et les réseaux de neurones artificiels. Nous nous attarderons sur les pistes suivantes : Le Backtracking et les algorithmes probabilistes (Méthode de Monté Carlo).

3.4.1 Le backtracking

Le backtracking est l'une des méthodes les plus simples et intuitives pour résoudre un sudoku. Son principe fondamental consiste à explorer systématiquement les différentes possibilités de remplissage des cases jusqu'à ce qu'une solution valide soit trouvée. Cela le rend adapté aux cas où les sudokus sont de complexité modérée ou inférieure. L'implémentation

de base de l'algorithme de Backtracking est relativement simple à comprendre et à programmer. Il repose sur un processus récursif où chaque étape consiste à remplir une case vide avec une valeur possible, puis à passer à la case suivante. Cette simplicité en fait un choix attrayant pour les amateurs de sudokus et les débutants en programmation. Cependant,

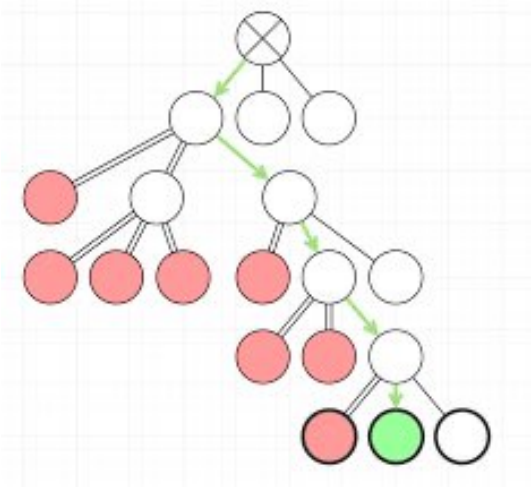

FIGURE 7 – Représentation du backtracking

les limites de l'algorithme de backtracking deviennent apparentes lorsque l'on se penche sur des sudokus plus complexes. Dans le pire des cas, la complexité temporelle de l'algorithme peut devenir exponentielle, ce qui signifie que le temps de résolution augmente considérablement à mesure que la complexité du sudoku augmente. Il peut devenir impraticable pour les sudokus très difficiles. De plus, l'algorithme de Backtracking ne dispose pas de mécanismes d'optimisation avancés pour éliminer les choix redondants ou accélérer la recherche. Il continue de tester des valeurs même lorsqu'il est évident qu'une solution n'existe pas, ce qui peut entraîner une perte de temps considérable, en particulier sur des sudokus invalides. La nature récursive de l'algorithme de Backtracking peut également poser des problèmes en termes d'efficacité d'exécution simultanée. Il n'est pas propice à l'utilisation de techniques de parallélisation, telles que les pipes et les forks, pour exploiter pleinement les capacités de traitement parallèle des systèmes modernes. Enfin, l'algorithme de Backtracking peut être gourmand en terme d'espace mémoire. La récursivité implique la création de nombreuses piles d'appel, ce qui peut entraîner une utilisation significative de la mémoire, en particulier pour les sudokus de grande taille. En somme, l'algorithme de Backtracking est un choix solide pour résoudre des sudokus plus simples aux moins complexes, mais il atteint ses limites pour les sudokus trop difficiles. Dans de tels cas, d'autres approches, telles que les algorithmes de recherche heuristique, les techniques de résolution par force brute informatique ou

les méthodes de résolution par contrainte, peuvent être plus adaptées pour obtenir des performances optimales.

3.4.2 Méthode de Monte Carlo et algorithmes probabilistes

L'algorithme de Monte Carlo est une méthode probabiliste puissante utilisée pour résoudre une grande variété de problèmes complexes, y compris la résolution de sudokus. Bien que cette méthode offre certains avantages, elle présente également des inconvénients importants qui méritent d'être pris en compte. L'algorithme de Monte Carlo est réputé pour sa simplicité et sa flexibilité. Il se base sur la génération aléatoire de solutions potentielles et l'évaluation de leur validité. Cette approche probabiliste peut être utilisée pour résoudre des sudokus en explorant un large éventail de combinaisons possibles, ce qui en fait une méthode appropriée pour les sudokus de toutes complexités. Cependant, l'un des inconvénients majeurs de l'algorithme de Monte Carlo est qu'il ne garantit pas la convergence vers une solution. Il s'agit d'une méthode stochastique, ce qui signifie qu'elle peut ne pas converger vers une solution dans un délai raisonnable, voire jamais. Cela peut entraîner des temps d'exécution imprévisibles, en particulier pour les sudokus complexes. De plus, l'algorithme de Monte Carlo dépend fortement de la qualité de la génération aléatoire. Des générateurs de nombres aléatoires de faible qualité peuvent introduire des biais et des erreurs dans le processus de résolution, compromettant ainsi la précision des résultats. En outre, l'algorithme de Monte Carlo peut nécessiter un grand nombre d'itérations pour obtenir une solution raisonnablement précise, ce qui peut entraîner une utilisation intensive des ressources informatiques. Une autre limitation importante réside dans le fait que l'algorithme de Monte Carlo ne profite pas pleinement de la connaissance préalable des contraintes du sudokus, contrairement à d'autres approches qui utilisent des techniques de contraintes pour éliminer les choix inutiles, l'algorithme de Monte Carlo explore des chemins inutiles, ce qui peut entraîner des temps d'exécution plus longs. En résumé, l'algorithme de Monte Carlo est une méthode intéressante pour résoudre des sudokus en utilisant des approches probabilistes. Il offre une grande flexibilité, mais il peut être imprévisible en terme de temps d'exécution, il dépend de la qualité de la génération aléatoire et il ne garantit pas la convergence vers une solution. Pour des sudokus complexes, d'autres méthodes, telles que le Backtracking, les algorithmes de recherche heuristique ou les techniques de résolution par contrainte, peuvent être préférées pour obtenir des résultats plus fiables et plus rapides.

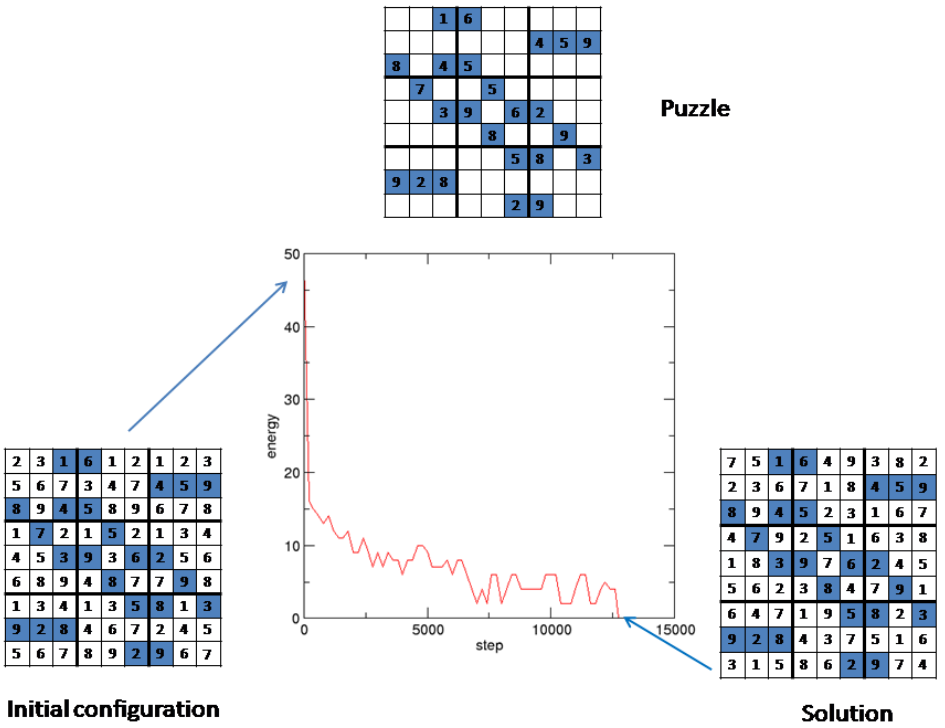

FIGURE 8 – Corrélation entre l'énergie et le stade de la résolution du sudoku selon la méthode de Monte Carlo

3.4.3 Choix final

Le choix final sera porté sur du backtracking amélioré, essayant de limiter au maximum le nombre d'appels récursifs. En raison de l'incapacité totale de faire fonctionner une implémentation de Monte Carlo, nous n'avons pas pu le choisir, bien que le principe soit plébiscité plus intéressant et original par le reste du groupe.

4 Conclusion et Futur

Nous avons remarqué de grandes difficultés a faire tout ce que nous voulions mettre en place. Surtout au niveau du pré-traitement de l'image. Nous avons aussi une multitude de moyens possibles, leurs avantages et leurs inconvénients, utilisables pour résoudre un problème aussi simple que la résolution d'un sudoku à travers une photo. Malgré beaucoup de soucis de parcours, nous avons su réussir à persévérer pour avancer dans le projet même si celui-ci a été intraitable avec nous.
Pour la prochaine soutenance, nous avons chacun des objectifs précis : l'apprentissage et la reconnaissance des chiffres de la grille de sudoku par le réseau de neurones pour Léo. Une application qui nous permettra d'avoir une interface graphique pour Grégoire et Nathan. Et un affichage des nouveau chiffres par Dardan. Le reste des tâches seront attribuées selon l'avancement du projet.