SORBONNE UNIVERSITÉ, UFR DE PHYSIQUE
Master 1 : Physique fondamentale et applications

Projet IA : Le Modèle d'Ising

Intelligence artificielle pour la physique

A. Cremel-Schlemer (3800159)
G. Carvalho (28709594)
M. Panet (28705836)

17 décembre 2023

TABLE DES MATIÈRES

TODO LIST

INTRODUCTION

Le modèle d'Ising est un modèle de physique statistique introduit par Rudolph Peierls, Wilhelm Lenz et Ernst Ising dans les années 1920. Malgré sa remarquable simplicité, ce modèle permet de mettre en évidence un comportement de transition de phase.

Sous sa formulation historique, le modèle d'Ising permet d'étudier la transition de phase paramagnétique/-ferromagnétique d'un matériau. Certains matériaux, appelés ferromagnétiques, possèdent une aimantation intrinsèque, comme les aimants. Cette propriété découle d'interactions à courte portée entre les spins d'un cristal. Cependant, le physicien Pierre Curie a découvert en 1895 que ces matériaux perdent leur propriété magnétique au-dessus d'une température caractéristique appelée température de Curie, notée T_C.

Il existe également des matériaux paramagnétiques, qui ne sont pas magnétiques à l'origine mais le deviennent en présence d'un champ magnétique extérieur puissant. En l'absence de champ magnétique extérieur, les spins de ces matériaux sont désordonnés, conduisant à un moment magnétique moyen nul, rendant le matériau non magnétique. Sous l'action d'un champ magnétique extérieur, ces moments se polarisent, rendant le matériau magnétique.

L'hamiltonien du modèle d'Ising prend donc en compte ces deux propriétés, avec une partie relative à l'interaction ferromagnétique à courte portée et une autre relative à l'interaction de chaque spin du cristal avec un champ magnétique extérieur. Dans la plupart des modèles numériques, on travaille sans champ extérieur.

La généralité du modèle d'Ising, avec ses interactions locales et l'interaction individuelle avec une force extérieure, transcende la physique et permet de décrire qualitativement et parfois quantitativement une grande variété de situations (paramagnétisme, gaz réticulaire, agents économiques, modèles écologiques, analyse et traitement de l'image). Il a même donné lieu à un modèle plus général : le modèle de Potts, qui généralise l'aspect binaire du modèle d'Ising lié à la binarité des spins.

À travers notre exposé sur l'intelligence artificielle, nous allons nous intéresser à un jeu de données constitué de 16,000 images issues d'une simulation numérique du modèle d'Ising 2D. Chaque image est associée à une température caractéristique allant de 0.25 à 4 (sans unité) et à un label. Le label renseigne sur la phase dans laquelle l'image se trouve. La *phase ferromagnétique* est une phase où les spins sont globalement tous alignés ou forment des îlots de spins alignés ($|M| > 0$). Le label *phase paramagnétique* rend compte du régime où les spins sont globalement désordonnés ($M = 0$).

Nous allons explorer différentes méthodes pour voir quelles solutions d'intelligence artificielle nous permettent de déterminer, à partir d'une image donnée, si l'on est dans la phase paramagnétique ou ferromagnétique. Nous nous intéresserons également à la manière dont sont générées les données du modèle d'Ising. Nous chercherons à voir s'il est possible de prédire la température d'une image de manière fiable et même s'il est possible de générer de nouvelles données sans faire appel à une simulation numérique coûteuse.

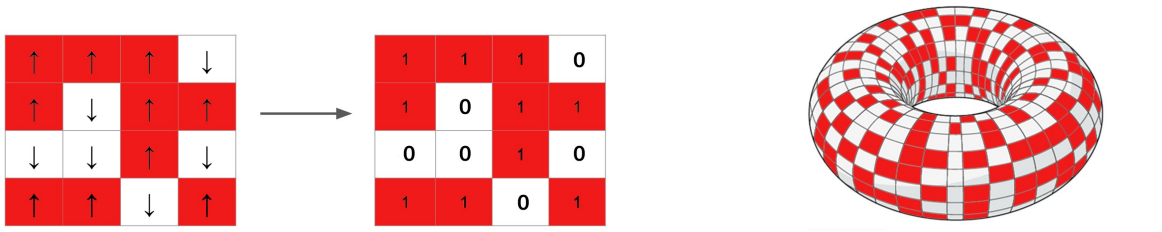
1 GÉNÉRATION DE DONNÉES

Afin de mieux cerner le problème initial, nous avons recréé la simulation du modèle d'Ising 2D pour des images de dimensions 40×40, comme celle présente dans le dataset.

1.1 Théorie, Hypothèses et Hamiltonien

Le modèle d'Ising 2D est une approche numérique du problème posé en introduction, où l'on fait l'hypothèse d'un cristal ferromagnétique plat et de maille carrée pour lequel chaque atome est associé à un pixel d'une image.

Chaque spin est représenté soit par un 0, soit par un 1, où le 1 correspond à un spin "Up" et le 0 à un spin "Down". Afin d'améliorer la validité du modèle et de surmonter les problèmes potentiels aux extrémités des images, des conditions aux bords périodiques sont utilisées. Concrètement, cela revient à considérer la surface d'un tore que l'on aurait déplié (cf. Figure 1.1).



(a) Les spins sont soit Up = 1, soit Down = 0

(b) Conditions aux bords périodiques

FIGURE 1.1 – Représentation du modèle d'Ising 2D

Le hamiltonien du problème est donné par l'équation :

$$H = -J\sum_{i,j}\sigma_i\sigma_j - h\sum_i \sigma_i$$

où $\sigma_i = \pm 1$. Cela dit, comme nous travaillons avec des 0 et des 1, il y a quelques manipulations à réaliser sur l'hamiltonien pour transformer les opérations de sorte que σ_i puisse prendre soit la valeur 0 soit la valeur 1.

Comme nous cherchons à mettre en évidence la façon dont un matériau ferromagnétique perd brutalement sa magnétisation en fonction de sa température, il devient clair que l'on ne s'intéresse pas à la deuxième partie de l'hamiltonien qui nous renseigne sur la manière dont les spins s'alignent avec un champ magnétique extérieur.

Il ne nous reste donc qu'à traiter :

$$H = -J\sum_{i,j}(2\sigma_i - 1)(2\sigma_j - 1)$$

Avec $\sigma_i \in \{0, 1\}$

Que signifie cet Hamiltonien ? C'est un Hamiltonien de couplage entre spins. Il est responsables du ferromagnétisme du matériau. À priori, pour respecter la physique du système que l'on cherche à étudier, ce couplage ne se réalise qu'à courte portée, entre particules identiques, ici le spin. Les spins "plus proches voisins" subissent alors une interaction d'échange à courte portée $\pm J$. Par simplicité et pour garder l'aspect très général de l'algorithme, on choisit de travailler avec des constantes adimensionnées de sorte que : $\frac{J}{K_B} = 1$.

1.2 Agitation thermique et probabilité

Nous allons maintenant nous approcher d'un des spins de notre modèle. Situé au centre, ses plus proches voisins forment une croix suisse autour de lui. Calculons maintenant son Hamiltonien local :

$$H_{local} = E_{ij} = (2\sigma_i - 1)\sum_j (2\sigma_j - 1) = (2\sigma_i - 1) \times \Big[2 \times (R + L + U + D) - 4\Big]$$

avec :

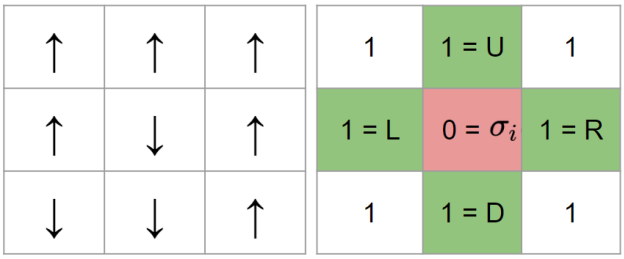


FIGURE 1.2 – Hamiltonien local

Partons d'un système idéal où tous les spins sont alignés. En raison de l'agitation thermique, un spin aléatoire du cristal inverse son sens. La variation d'énergie associée à ce changement est donnée par :

$$\Delta E = E_f - E_i$$

Et la probabilité qu'un tel changement s'opère est proportionnelle au poids de Boltzmann :

$$P(\Delta E) \propto e^{-\beta \Delta E} \propto e^{-\frac{\Delta E}{T}}$$

L'idée est la suivante : l'ordinateur va piocher une case au hasard, calculer le ΔE associé. Ce ΔE ne peut prendre que 5 valeurs possibles :

n voisin de sens opposé	ΔE
0	8
1	4
2	0
3	-4
4	-8

Dans le cas où $\Delta E \in 0, -4, -8$, l'énergie est minimisée, et donc le changement s'opère naturellement : l'ordinateur modifie le pixel d'état. Dans le cas inverse, c'est-à-dire lorsque $\Delta E \in 4, 8$, un nombre aléatoire x est tiré selon une distribution uniforme entre 0 et 1 et est comparé à $e^{-\frac{\Delta E}{T}}$. Si $x < e^{-\frac{\Delta E}{T}}$, alors l'ordinateur modifie le pixel d'état ; sinon, il ne le fait pas.

Nous remarquons que plus T est élevée, plus la probabilité de transition est importante.

On peut alors lancer la simulation. Pour une température fixée, nous réalisons un certain nombre de tentatives de changement de spin. Le système va se relaxer jusqu'à atteindre un équilibre, où la moyenne des spins composant l'image est stable après chaque tentative successive de changement de l'état d'un spin de l'image.

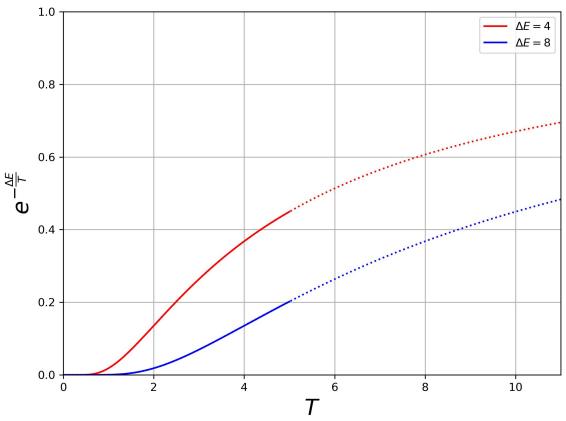


FIGURE 1.3 – probabilité de transition en fonction de la température

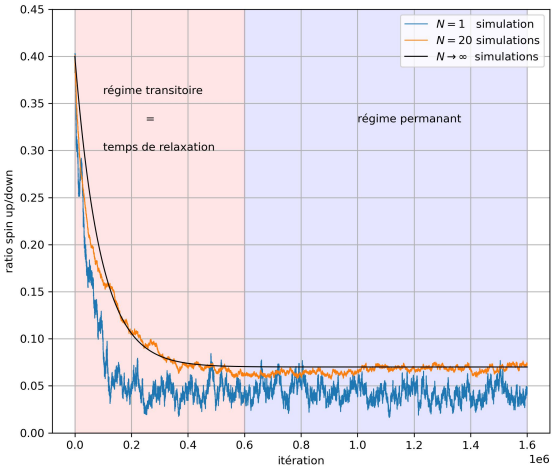


FIGURE 1.4 – Évolution du ratio spin up/spin down pour $T = 2$ et 1,6 million d'itérations.

1.3 Stabilité, cohérence de la solution et considérations techniques

Pour chaque température, le temps de relaxation du système est plus ou moins long et il est également influencé par la composition du système de départ. Dans le cas où l'on aurait une situation initiale avec que des spins alignés, le temps de relaxation vers l'équilibre pour T proche de 0 est très rapidement trouvé car le système est déjà à l'équilibre. Le matériau est bien ferromagnétique : $\frac{|M|}{M_{max}} > 0$.

En revanche, pour T proche de 0 et un système de départ homogène avec autant de 0 que de 1, le temps de relaxation est plus lent et fait apparaître de nouveaux états d'équilibre. Les spins peuvent tous s'aligner dans un sens ou dans un autre. Il existe une autre formation, de moment magnétique total nul où il y a coexistence d'une bande up et d'une bande down. Avec l'analogie du tore, c'est comme si on avait trempé un donut dans du chocolat jusqu'à la moitié. C'est une situation qui se rapproche de la situation dite d'antiferromagnétisme (lien).

Pour éviter des situations non souhaitées, nous initialisons la simulation avec un ratio de 0.25% ou 0.75% de 1. Cela facilite la convergence vers un état d'équilibre dans les basses et les hautes températures et limite l'apparition de figures intermédiaires qui ne correspondraient pas à la physique que l'on cherche à mettre en avant (ferromagnétisme).

Le temps de calcul avec un algorithme non optimisé écrit en Python est lent. Générer 160,000 nouvelles

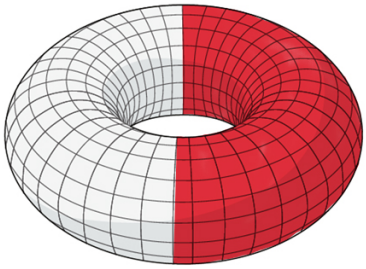


FIGURE 1.5 – Artefact à basse température $M = 0$

données est très coûteux. Cela est lié à plusieurs choses. Premièrement, Python est un langage de haut niveau, intelligible par un humain mais très éloigné d'un langage machine comme l'assembleur où la nature des données est très spécifiquement choisie et traitée. En Python, il existe donc une procédure de "traduction" de l'humain vers la machine qui est coûteuse et conduit à des performances réduites. Il est possible d'optimiser le programme avec des principes généraux de réduction du nombre de calculs à effectuer pour réaliser une tâche (calcul en amont de certaines variables, etc.).

Il est aussi possible d'écrire ce même code dans un langage compilé, comme le C++, qui est un langage dit "typé", c'est-à-dire que l'utilisateur doit faire un effort pour indiquer à l'ordinateur la nature des variables avec lesquelles il travaille pour faciliter le travail de "traduction" vers un langage machine. Cela évite "l'interprétation" au moment de la compilation du programme (la "traduction") et le programme est donc généralement plus rapide.

Il est possible, en Python et en C++, d'accélérer le programme par la parallélisation de certains calculs. C'est-à-dire que l'ordinateur va exécuter la même tâche en parallèle pour gagner du temps, un peu comme si nous, humains, pouvions recopier simultanément un texte avec la main gauche et la main droite sur deux feuilles distinctes. Ici, l'ajout de nouveaux points se fait en parallèle.

Mais il y a un problème : si en parallèle on modifie des pixels adjacents, il risque d'y avoir un conflit dans le calcul de l'énergie. Ce problème n'est pourtant pas très important dans une image 40 par 40 dès lors que le nombre d'exécutions en parallèle n'est pas trop grand. En effet, pour 16 tâches en parallèle, $16 \times 9 = 144$ pixels sont manipulés en parallèle, ce qui représente une fraction de 9% de l'image. En approximation, on peut dire que les erreurs en énergie sont suffisamment négligeables. Nous générons les données suivantes avec cette hypothèse et cela semble plutôt correct.

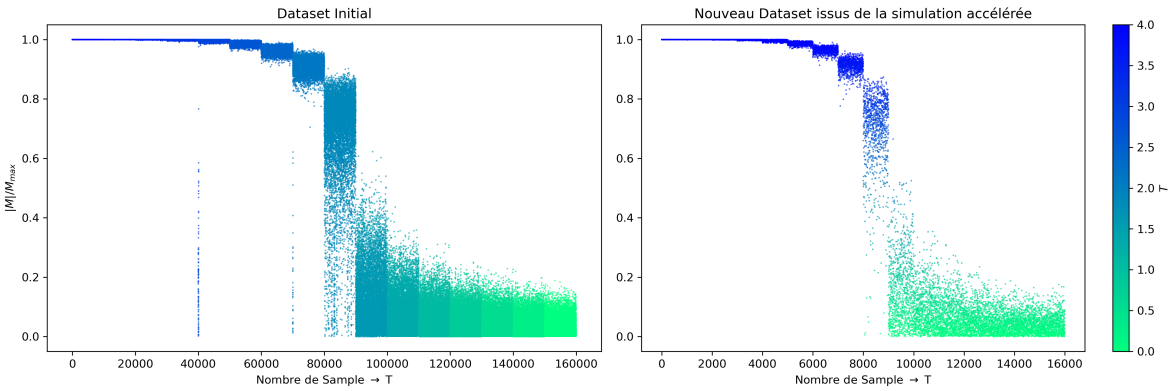


FIGURE 1.6 – Nouvelles figures et comparaison avec le dataset

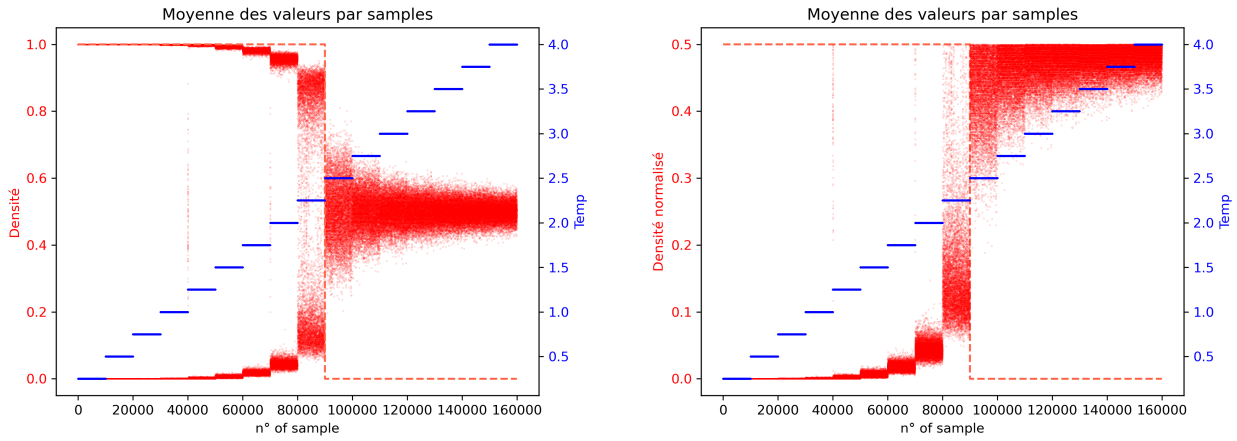
Pour prendre en compte les probabilités correctement avec la parallélisation et pour accélérer le calcul par des opérations très bas niveau, il faut se tourner vers l'algorithme de Metropolis-Hastings mis en place en 1953.

2 Pré-traitement des données

Maintenant que nous avons généré nos données, nous devons nous faire une idée de la forme de nos données afin de pouvoir les traiter de la meilleure façon possible. Les données fournies au départ sont des vecteurs de taille 1600 contenant des 1 et des 0. Ces vecteurs représentent des configurations de spins se trouvant sur une grille 2D de taille 40×40. Le dataset original est composé de 10000 configurations de spins pour 16 températures différentes comprises entre 0.25 et 4.00 avec un pas de 0.25. Avec ces configurations, un label est associé à chaque configuration. Ce label est une valeur binaire qui nous indique la phase dans laquelle se trouve le système. De plus, nous avons généré un second jeu de données avec X configurations de spins pour Y températures comprises entre 0.25 et 4.00. Ce second jeu de données nous permettra de tester nos modèles sur des données avec des températures différentes de celles du jeu de données original. On pourra ainsi voir si nos modèles sont capables de se séparer des températures discrètes du jeu de données original. Comme on peut le voir sur la figure 2.1a, nos données forment un ensemble bruité mais il apparaît une symétrie par rapport à l'axe horizontal. En effet, à basse température, les spins sont majoritairement alignés de la même façon mais de manière aléatoire en + ou −. Cette symétrie de nos données peut poser un problème à nos modèles qui auront dû apprendre à faire la différence entre deux configurations opposées mais équivalentes. Pour éviter ce problème, nous allons symétriser nos données en inversant les spins de toutes les configurations qui ont une moyenne de spin négative. Ainsi, on se retrouve avec des données symétriques par rapport à l'axe horizontal comme on peut le voir sur la figure 2.1b.

change number

change number



(a) Données brutes

(b) Données symétrisées

Figure 2.1

Dans la partie suivante, nous allons entraîner certains modèles spécifiques sur la densité de spin symétrisée. Dans ce cas, nous allons aussi normaliser afin de rendre les modèles plus performants. Pour cela, nous allons utiliser la méthode *StandardScaler* de la librairie *sklearn* qui permet de centrer et réduire les données. Cette méthode soustrait la moyenne et divise par l'écart-type. Ainsi, on se retrouve avec des données centrées en 0 et de variance 1.

Il nous faut maintemant séparer nos données en deux ensembles : un ensemble d'entraînement et un ensemble de test. Pour cela, nous allons utiliser la méthode *train_test_split* de la librairie *sklearn*. Cette fonction nous permet de préciser le taux de répartition des données entre l'ensemble d'entraînement et l'ensemble de test. Nous allons répartir 80% des données dans l'ensemble d'entraînement et 20% dans l'ensemble de test. De plus, cela nous permet de mélanger les données avant de les répartir afin d'avoir dans les deux ensembles des données représentatives de notre jeu de données originel.

Afin d'appliquer exactement la même transformation sur les données de test, nous allons créer un pipeline qui va appliquer la méthode de symétrisation puis la méthode de normalisation. Ainsi, on pourra appliquer le pipeline sur les données de test sans avoir à les modifier. Finalement, certains modèles seront plus réceptifs à des données présentées sous forme de vecteur de taille 1600, d'autre sous forme de matrice de taille

40×40. Pour éviter de devoir modifier les données à chaque fois, nous allons créer un pipeline qui va transformer les données en matrice si nécessaire et appliquer les autres méthodes de pré-traitement expliquées ci-dessus.

3 Modèles classiques

3.1 Analyse par la magnétisation

Notre première approche de ce problème est de considérer la magnétisation comme une fonction de la température. Cette approche nous permet de réduire grandement la complexité du problème. En effet, nous n'avons plus qu'une seule variable à considérer : l'état moyen des spins. Dans cette partie, nous allons donc essayer de prédire la température à partir de la magnétisation. Il suffit ensuite de comparer la valeur prédite à la valeur de la température critique pour déterminer la phase du système.

Avant tous, nous allons établir un modèle naif qui va nous servir de référence. Ce modèle va simplement renvoyer la valeur moyenne de la température du jeu de données. Ainsi, on pourra comparer les performances de nos modèles avec ce modèle naif. On obtient une erreur quadratique moyenne de 1.32.

Du fait du bruit de nos données et du faible nombre de températures distinctes, nous devons porter une attention particulière au sur-apprentissage de nos modèles. Prenant en compte le grand nombre de données, nous sommes partis sur un modèle de foret aléatoire. Ce modèle est très robuste et permet de limiter le sur-apprentissage. De plus, il est très rapide à entraîner et à tester. Au niveau des hyperparamètres, nous avons limité la profondeur des arbres à 5 et nous avons aussi composé la foret de 100 arbres. De plus, la méthode de *Bootsrap* est activée. Cette méthode permet de créer des sous-ensembles de données de la taille de l'ensemble de données originale. Ainsi, on peut entraîner plusieurs arbres sur des données différentes et les combiner pour obtenir un modèle plus robuste. On peut donc limiter le sur-apprentissage tout en gardant un modèle performant. Les résultats obtenus sont présentés sur la figure 3.1a. La figure 3.1a possède deux objets : La densité de points de la moyenne des spins symétrisée par température calculé en utilisant la méthode de *KDE* (*Kernel density estimation*) et la prédiction du modèle de foret aléatoire.

Avec une MSE de 0.108, on obtient un modèle qui est 12 fois plus performant que le modèle naif. Cependant, on peut voir que le modèle a du mal à distinquer les hautes températures. En effet, les valeurs prédites par ce modèle sont toujours inférieures à 3.5. Cela peut s'expliquer par le fait que, à haute température, la densité est quasiment nulle. Ainsi, lorsque le modèle va essayer de prédire une configuration de spin à haute température, il va prédire la température correspondant plus ou moins à la moyenne des températures capable de produire une configuration de spin similaire. Cela apparaît clairement sur la figure 3.1a.

Nous pouvons, maintemant, utiliser la prédiction en température de notre modèle pour déterminer la phase du système. Pour cela, nous allons comparer la valeur prédite à la valeur de la température critique. Si la valeur prédite est supérieure à la valeur de la température critique, alors le système est dans la phase paramagnétique. Sinon, il est dans la phase ferromagnétique. Théoriquement, la température critique d'un modèle d'Ising 2D est de 2.269. On retrouve les résultats de cette analyse sur la figure 3.1b. Notre modèle est capable de prédire la phase du système avec une précision de 98.8%. On a donc un modèle performant.

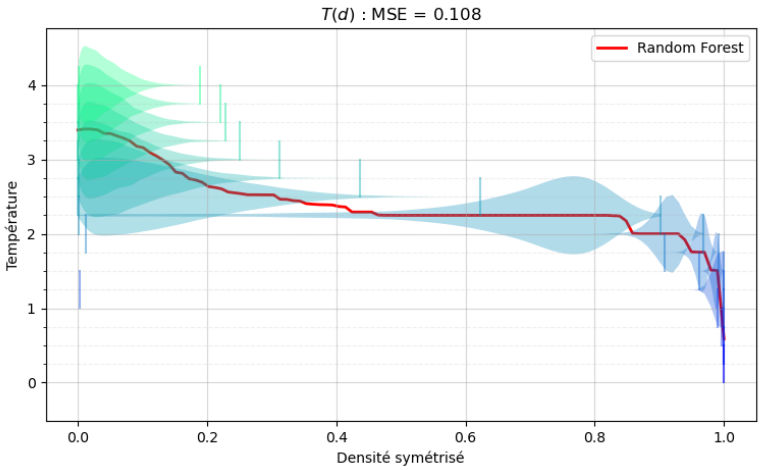
cite une source

3.2 Analyse par l'état individuel des spins

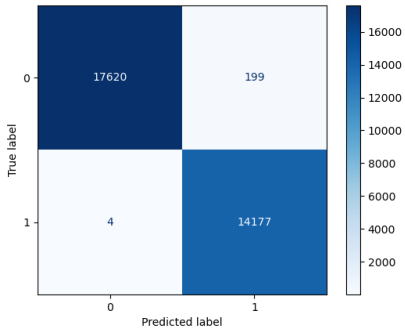
4 Réseaux de neurones

4.1 Objectif

Dans le cadre de ce projet, notre objectif final était de développer un modèle performant capable de classer des matrices de taille 40x40 en deux catégories distinctes. Comme dans le modèle d'Ising l'état d'un spin



(a) Résultats du modèle



(b) Matrice de confusion

FIGURE 3.1 – Modèle de foret aléatoire

est dépendant de ces plus proches voisins, le plus naturel était de créer un réseau neuronal convolutif car il prend en compte les corrélations locales entre pixels voisins. La performance de notre classification est alors dépendante de l'architecture de notre modèle, de la manière dont il extrait et apprend les caractéristiques pertinentes des matrices en entrée.

Dans cette partie, nous allons analyser l'architecture du modèle qu'on a créé. On va discuter pourquoi on a choisi chaque couche et quel rôle elles jouent dans notre modèle. On analysera ensuite nos étapes d'entraînement puis on discutera les méthodes que nous avons utilisé pour évaluer nos performances.

4.2 Préparation des données

section à complété

4.3 Architecture du modèle

L'ensemble de l'architecture repose sur des principes de convolution, pooling et de couches entièrement connectées. La combinaison de ces différentes couches crée une architecture cohérente et un modèle performant.

Notre modèle commence par une première couche de convolution 2D, qui utilise 16 filtres de taille 5x5. Cette couche veut extraire des caractéristiques de bas-niveau, soit des bords ou des formes simples présentes dans les données. Cette première couche de convolution est suivie d'une couche de max pooling qui permet, en faisant un sous-échantillonnage, de compresser l'information que l'on veut faire passer à la couche suivante.

On applique ensuite à nouveau ce processus : on a une deuxième couche de convolution 2D a 32 filtres de taille 3x3 et une nouvelle couche de max pooling. Avec cette deuxième couche de convolution, on souhaite capturer des motifs plus complexes et abstraits. On a augmenté le nombre de filtres de la première couche de convolution car on veut augmenter la capacité du modèle : chaque filtre pourra se spécialiser dans la détection de certaines caractéristiques et donc le modèle apprendra plus de relations complexes. Avec la couche de max pooling, on souhaite réduire davantage la dimension spatiale. Finalement, on utilise une dernière couche de convolution 2D avec 64 filtres de taille 3x3, suivie encore d'une autre opération de max pooling.

Ensuite, on utilise une couche Flatten pour aplatir les caractéristiques et motifs extraits de nos couches de convolution pour réduire la dimension de notre matrice. Il s'agit d'une opération assez simple avant d'introduire des couches entièrement connectées Dense. On a utilisé trois couches consécutives : la première

avec 128 neurones, la deuxième avec 64 neurones puis la troisième avec 128 neurones. Ces trois couches ont comme objectif d'apprendre les complexités des caractéristiques extraites, afin de réaliser l'objectif de notre modèle. Notre deuxième couche Dense a moins de neurones pour réduire les dimensions et additionner encore un peu de complexité à notre modèle, afin de tout bien représenter. Cette architecture que nous avons utilisée permet une certaine flexibilité au modèle et de s'adapter à la tâche de la classification binaire.

Finalement, l'architecture de notre modèle se termine par une couche de sortie avec une seule cellule, activée à la fonction d'activation sigmoïde. Cela nous permet d'obtenir à la sortie de notre modèle une probabilité que notre sortie appartienne à la classe positive, ce qui correspond bien à ce qui est attendu lors d'une classification binaire.

L'architecture de notre modèle est présentée sous forme d'image dans la figure 4.1

4.4 Entraînement du modèle

La procédure d'entraînement consiste à entraîner le modèle à réaliser une classification binaire : identifier s'il y a eu une transition de phase par rapport aux matrices qui décrivent notre système. Avant l'entraînement, nous avons compilé notre modèle avec l'algorithme optimiseur Adam et on a utilisé binary crossentropy comme fonction de coût, une fonction de coût très efficace pour les tâches de classification binaire qui mesure la différence entre les prédictions binaires et les vrais résultats. Finalement, l'entraînement de notre modèle a duré que 11 epochs, avec un batch size de 128. Notre entraînement a duré que 11 epochs pour éviter le sur-apprentissage, puisque nous avons utilisé la méthode de l'arrêt anticipé (early stopping), avec une patience de 5. L'entraînement du modèle a été fait avec les 160 000 matrices que l'on nous as fournies et a pris que 40 secondes.

Pendant l'entraînement, l'erreur d'entraînement et l'erreur de validation sont enregistrés pour chaque epoch. Cela nous permet de suivre l'entraînement de notre modèle et de détecter des signes de sur-apprentissage ou sous-apprentissage. En suivant cette approche et cette architecture, nous sommes parvenus à un modèle très efficace, dont nous allons étudier les performances maintenant.

4.5 Evaluation des performances du modèle

L'objectif principal du modèle est de capturer des motifs complexes qui permettront au modèle d'apprendre les complexités de nos matrices données en entrée et puis faire des prédictions sur à quelle classe appartient chaque matrice. Avec l'enregistrement de l'entraînement et la possibilité de faire des essais avec notre jeu de test, on peut évaluer notre modèle.

Tout d'abord, on peut analyser la performance de notre modèle en prenant en compte l'erreur de la fonction coût sur l'ensemble de données de validation, qui est de seulement 0.0084. Cela indique que le modèle est très performant : il a atteint un niveau très bas de perte sur les données de validation et a une très bonne capacité de se généraliser à de nouvelles données. On peut aussi vérifier la précision (accuracy) de notre modèle sur les données de test : 0.99. Notre modèle est capable de classer correctement 99% des échantillons dans l'ensemble de test. Cela renforce à nouveau l'idée que notre modèle a une très bonne capacité à généraliser et qu'il est très adéquat à la classification binaire. Nous avons aussi calculé une autre métrique intéressante : la sensibilité, mesurant la capacité du modèle à identifier tous les exemples positifs réels dans l'ensemble de données et nous trouvons une valeur de 0.99, indiquant une excellente capacité à mesurer des vrais positifs.

compare aux autre modèles

Ensuite, nous pouvons étudier l'entraînement de notre modèle. Dans la figure 4.2, nous observons l'évolution de l'erreur de la fonction de coût des données d'entraînement et de validation au cours de l'entraînement (graphe en haut). On remarque que les courbes représentant ces données sont très proches dès la première epoch jusqu'à l'epoch 6 et puis commencent à diverger ce qui peut être un signe de sur-apprentissage. Cependant, cela peut être négligeable dû à la simple différence de 0.005 entre les deux

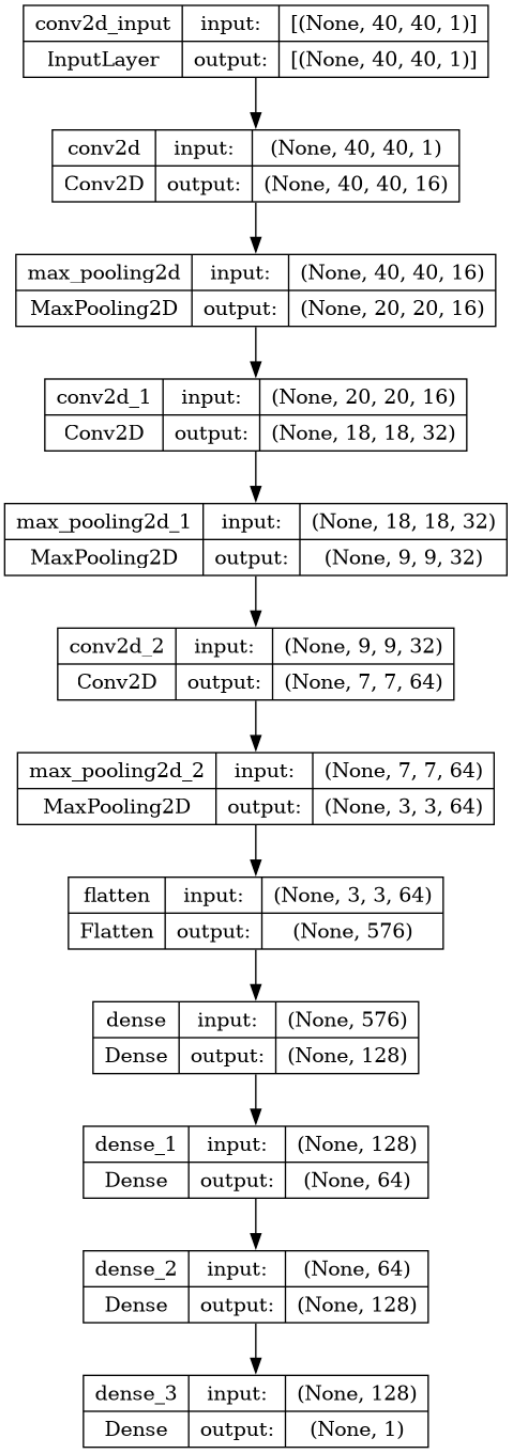


FIGURE 4.1 – Architecture du modèle

courbes. On souligne encore que l'évolution de l'entraînement est très légère, sans grande amélioration de l'erreur sur la fonction de coût. Encore dans la figure 4.2, on observe l'évolution de la précision au cours de l'entraînement (graphe en bas). On remarque que dès la première epoch jusqu'à la fin de l'entraînement, ces deux courbes sont toujours très proches autour de 0.998, ce qui confirme à nouveau excellente précision de notre modèle.

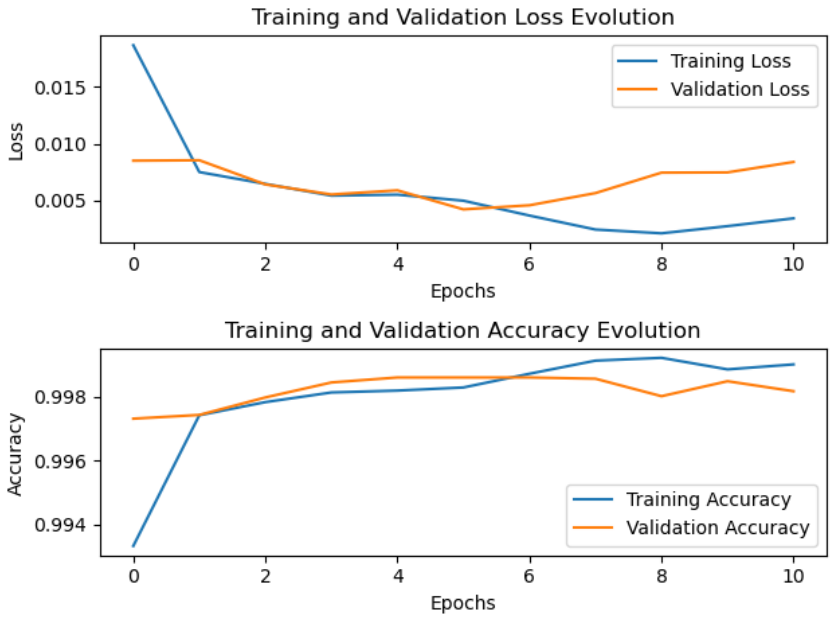
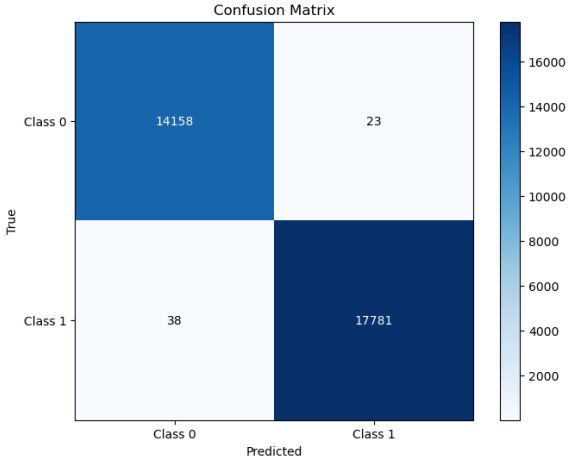
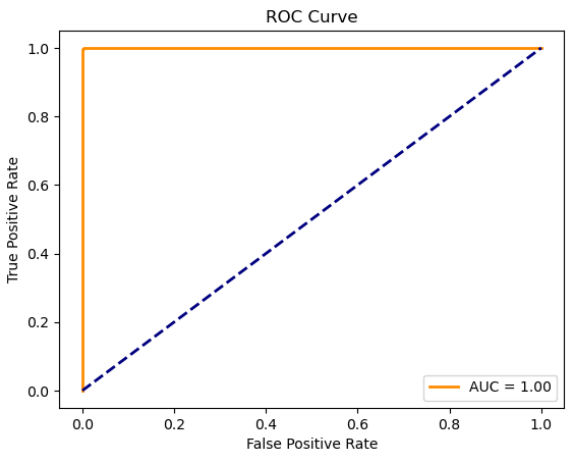


FIGURE 4.2 – Évolution de l'entraînement

De plus, nous pouvons établir la matrice de confusion de notre modèle, que nous pouvons observer à la figure 4.3a. La matrice de confusion résume le nombre de prédictions correctes et incorrectes faites par le modèle sur l'ensemble des données de test. On observe les vrais négatifs et les vrais positifs, représentés respectivement en haut à gauche et en bas à droite de la matrice de confusion. Ces nombres représentent les prédictions correctes du modèle, où il a réussi à identifier correctement les instances des deux classes, c'est-à-dire s'il y avait une transition de phase ou pas. Lors des prédictions sur le jeu de test, nous avons 14 158 vrais négatifs (nombre d'échantillons qui étaient réellement dans la classe 0 et correctement identifiés par le modèle) et 17 781 vrais positifs (classe 1 correctement identifiée). De plus, on peut encore observer les faux négatifs et les faux positifs, représentés respectivement en bas à gauche et en haut à droite de la matrice de confusion. Ils représentent les erreurs de classification du modèle. On a 36 faux négatifs et 23 faux positifs, ce qui est négligeable face au nombre de prédictions correctes. Le modèle créé est alors extrêmement efficace à prédire la transition de phase.



(a) Matrice de confusion



(b) Courbe ROC

Finalement, nous pouvons tracer la courbe ROC (Receiver Operating Characteristic), vu dans la figure 4.3b. Il s'agit d'un autre outil d'évaluation de la performance d'un modèle de classification binaire. Elle représente graphiquement la relation entre le taux de vrais positifs, connu aussi comme la sensibilité (ou recall) et le taux de faux positifs lorsqu'on fait varier le seuil de classification. On observe qu'on obtient un AUC (Area Under Curve) de 1.00, ce qui nous indique une performance exceptionnelle de notre modèle. Une valeur de 1.00 signifie une séparation parfaite des classes. Le modèle est très précis dans sa capacité à distinguer les exemples positifs des négatifs.

Dans cette partie, nous avons construit un réseau neuronal convolutif performant capable de classer efficacement nos données par rapport à l'existence ou pas d'une transition de phase. L'architecture du modèle choisie permet d'extraire les caractéristiques complexes de nos données et l'entraînement a été optimisée pour cette tâche de classification binaire. Les métriques de notre modèle tel que la précision, la sensibilité, la matrice de confusion et la courbe ROC montrent la robustesse de notre modèle. Nous avons alors créé un modèle à la fois précis et efficace, qui démontre une performance exceptionnelle dans le classement des matrices du modèle Ising.

CONCLUSION