جامعة مولاي إسماعيل
ⵜⴰⵙⴷⴰⵡⵉⵜ ⵎⵓⵍⴰⵢ ⵉⵙⵎⴰⵄⵉⵍ
UNIVERSITÉ MOULAY ISMAÏL

كلية العلوم
ⵜⴰⵖⵣⵓⵜ ⵏ ⵜⵣⵔⴼⵜ
FACULTÉ DES SCIENCES

Filière : Sciences de la Matière Physique

MÉMOIRE DE PROJET TUTORÉ

Présenté

Par:

Ayoub EL MHAMDI
youssef MADANE

UTILISATION DU DEEP LEARNING POUR IDENTIFIER LES NODULES PULMONAIRES CANCÉREUX SUR LES IMAGES DE TDM

Soutenu le **/07/2023 devant la Commission d'Examen :

Pr RAJAE Sebihi	**Encadrant**

REMERCIEMENTS

Nous tenons à remercier d'abord toutes les équipes pédagogiques de **la Filière Science de la Matière Physique** de la Faculté des Sciences à Meknès, ainsi que les professeurs ayant contribué activement à notre formation.

Nous profitons de cette occasion, pour remercier vivement notre Professeur **RAJAE SEBIHI** qui n'a pas cessé de nous encourager tout au long de l'exécution de notre Projet de Fin d'Études, ainsi que pour sa générosité et ses compétences en matière de formation et d'encadrement. Nous lui sommes reconnaissants pour ses aides et conseils précieux qui nous ont permis de mener à bien le présent projet.

Nos vifs remerciements vont aussi aux membres de jury pour avoir accepté de juger ce travail.

A la même occasion, nous voudrons également remercier chaleureusement nos parents qui nous ont toujours encouragés durant notre cursus de formation.

Enfin, nos vifs remerciements sont adressés à toutes ces personnes qui nous ont apporté leur aide précieuse et leur soutien inconditionnel. ■

TABLE DES MATIÈRES

RÉSUMÉ.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do.

INTRODUCTION GÉNÉRALE.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do. ■

RÉFÉRENCES BIBLIOGRAPHIQUES.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do. ■

Chapitre I.

APERÇU SUR DEEP LEARNING

I.1. Introduction

L'apprentissage profond est une branche de l'apprentissage automatique[1], qui est lui-même un domaine de l'intelligence artificielle. L'apprentissage profond permet de prédire ou d'analyser des données de haute dimension ou complexes, comme les images, les textes ou les sons, d'une manière similaire au cerveau humain. L'apprentissage profond utilise des réseaux de neurones artificiels multicouches[2] qui peuvent extraire les sens et les motifs cachés dans les données sans avoir besoin d'intervention humaine. Ainsi, l'apprentissage profond acquiert une grande capacité d'adaptation et d'évolution avec le changement de l'environnement.

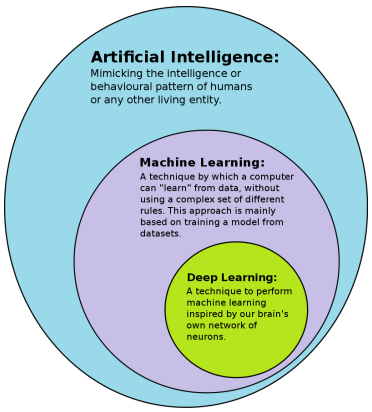

Figure 2: Comment l'apprentissage en profondeur est un sous-ensemble de l'apprentissage automatique et comment l'apprentissage automatique est un sous-ensemble de l'intelligence artificielle (IA)

Par conséquent, l'apprentissage profond se distingue de l'apprentissage traditionnel par le fait qu'il ne repose pas sur des règles ou des algorithmes prédéfinis, mais qu'il peut générer ses propres règles et algorithmes par essai et erreur. De plus, l'apprentissage profond peut surmonter certains des problèmes rencontrés par l'apprentissage traditionnel, tels que le bruit, le manque ou le changement des données.

Ainsi, nous voyons que l'apprentissage profond est un domaine récent et prometteur en informatique qui mérite l'attention et la recherche, et qui peut contribuer à résoudre de nombreux problèmes dans différents domaines tels que la traduction, la reconnaissance d'images et de sons, etc.

I.2. Fonctionnement de Deep Learning

Pour comprendre le principe de l'apprentissage profond, on peut utiliser des exemples de notre vie quotidienne. Lorsque nous voulons améliorer certains résultats, on change certains facteurs influençant ces résultats de manière cyclique, en se basant sur l'expérience et l'erreur. Par exemple, un vendeur de fruits essaie d'augmenter son revenu en changeant la quantité et les types de fruits offerts aux clients, en se référant aux ventes passées et actuelles. Il n'y a pas de règle fixe qui détermine la quantité de chaque fruit que le vendeur doit fournir, il doit donc expérimenter jusqu'à ce qu'il atteigne le point d'équilibre entre l'offre et la demande.

En apprentissage profond, on utilise une fonction mathématique appelée fonction coût pour mesurer la différence entre les résultats d'un modèle d'apprentissage et les résultats souhaités ou corrects. Puis on utilise une autre fonction appelée fonction optimisation pour ajuster la valeur de chaque cellule neuronale dans le réseau d'apprentissage afin de réduire la valeur de la fonction coût. Ces étapes sont répétées sur un grand ensemble de données jusqu'à ce que le modèle d'apprentissage soit capable d'accomplir les tâches demandées avec précision ou acceptabilité.

Cet exemple peut nous donner une idée qui nous aide à comprendre l'apprentissage en profondeur, mais il résume des concepts fondamentaux de l'apprentissage en profondeur tels que la fonction de coût ou la fonction de régression graduelle et l'optimisation[1], ce qui est clair pour nous dans des applications telles que la traduction automatique ou la vision par ordinateur.

Dans la traduction automatique, un système d'apprentissage en profondeur utilise un réseau neuronal pour convertir une phrase d'une langue à une autre. Ce système utilise une fonction de coût pour mesurer la différence entre la traduction générée et la traduction cible. Ensuite, il utilise un algorithme d'optimisation tel que la descente de gradient pour ajuster le poids de chaque cellule neuronale dans le réseau afin de minimiser la valeur de la fonction de coût. Ce processus est répété sur un grand ensemble de phrases à traduire jusqu'à ce que le système soit capable de générer des traductions précises et naturelles.

En vision par ordinateur, un système d'apprentissage en profondeur utilise des réseaux neuronaux artificiels pour extraire des informations et des perspectives à partir d'images et de vidéos. Certaines applications dans ce domaine sont[3]:

- La **classification d'images** consiste à attribuer une étiquette à une image ou une photographie entière.

Ce problème est également appelé « classification d'objets » et peut-être plus généralement « reconnaissance d'images », bien que cette dernière tâche puisse s'appliquer à un ensemble beaucoup plus large de tâches liées à la classification du contenu des images.

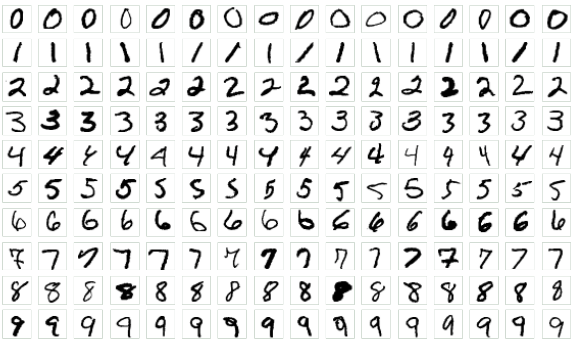

Figure 3: Un exemple de chiffres de la base MNIST.

Un exemple populaire de classification d'images utilisé est le jeu de données(dataset) MNIST.

- **Surveillance du contenu**: pour supprimer automatiquement le contenu non sécurisé ou inapproprié des archives d'images et de vidéos.
- **Reconnaissance faciale**: pour identifier l'identité des personnes ou extraire des caractéristiques de leur visage, telles que l'ouverture ou la fermeture des yeux, le port de lunettes ou de moustaches.
- **Violation du droit d'auteur**: pour supprimer le contenu volé ou détourné d'images ou de vidéos protégées par des droits d'auteur.

Ces applications sont rendues possibles grâce à l'utilisation de réseaux neuronaux profonds qui peuvent apprendre à partir d'un grand nombre d'exemples et extraire des caractéristiques complexes à partir des données.

I.3. Les réseaux de neurones artificiels

Les réseaux de neurones artificiels sont des modèles d'intelligence artificielle qui utilisent des cellules nerveuses artificielles pour convertir les entrées en sor-

ties. Chaque cellule nerveuse reçoit des signaux d'autres cellules et envoie des signaux à d'autres cellules. Chaque signal est ajouté à une valeur de poids qui détermine sa force et son importance. Chaque cellule nerveuse calcule la somme des signaux reçus et applique une fonction d'activation pour produire un signal de sortie.

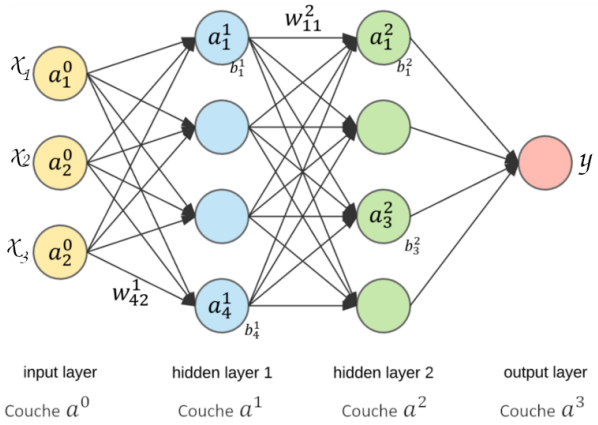

Figure 4: architecture d'un perceptron multicouche.

I.4. Exemple de Deep learning dans la pratique

pour savoir la relation entre les réseaux de neurones artificiels et la fonction linéaire, on utilise l'exemple de calcule de temperature precedent.

La fonction linéaire est un type de fonctions mathématiques qui décrit une relation simple entre deux variables, où un changement dans l'une entraîne un changement proportionnel dans l'autre. Par exemple, si nous avons une fonction qui convertit la température de Celsius en Fahrenheit, cette fonction sera linéaire, car chaque augmentation d'un degré Celsius entraîne une augmentation constante de la température en Fahrenheit.

La formule utilisée pour convertir la température de Celsius en Fahrenheit est [4] :

$$°F = \frac{9}{5}°C + 32$$

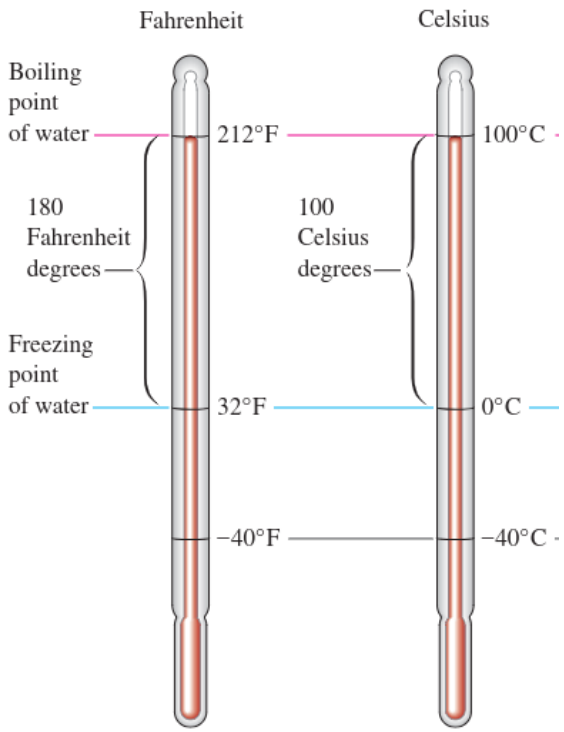

Figure 5: Relation enter deux échelles de température Celsius et Fahrenheit

Dans cette formule, nous pouvons définir deux facteurs principaux : le poids des entrées et l'ordonnée à l'origine. Le poids des entrées est un nombre qui multiplie la valeur de la variable indépendante (℃) pour déterminer son effet sur la valeur de la variable dépendante (°F). Dans ce cas, le poids des entrées est 1.8. L'ordonnée à l'origine est un nombre qui est ajouté au produit du poids des entrées par la valeur de la variable indépendante pour déterminer la valeur de la variable dépendante lorsque la variable indépendante est égale à zéro. Dans ce cas, l'ordonnée à l'origine est 32.

Ces exemples montrent comment utiliser une fonction linéaire pour convertir la température de Celsius en Fahrenheit. Mais comment utiliser une fonction linéaire pour comprendre le concept de coût et d'optimisation dans l'apprentissage profond ?

Dans l'apprentissage profond, nous utilisons des fonctions linéaires pour produire des résultats basés sur un ensemble d'entrées. Chaque entrée a un poids qui ajuste la mesure de son impact sur le résultat produit. Ensuite, nous utilisons une fonction de coût pour mesurer la différence entre les résultats produits et les résultats cibles. Ensuite, nous utilisons un algorithme d'optimisation[5][6] pour ajuster les poids des entrées afin de réduire la valeur de la fonction de coût.

Comme première étape dans l'apprentissage profond, nous devons collecter un ensemble de données qui forment des paires d'entrées et de résultats cibles.

Dans l'exemple de conversion de la température de Celsius en Fahrenheit, nous utilisons un tableau avec des mesures différentes de température dans les deux systèmes comme des paires d'entrées et de résultats cibles :

℃	℉
-40	-40
-20	-4
0	32
20	68
40	104
60	140
80	176
100	212

Ce tableau nous permet de comparer la valeur de la température en Celsius avec sa valeur correspondante en Fahrenheit. Mais que se passe-t-il si nous voulons convertir une valeur qui n'est pas dans le tableau ? Pouvons-nous trouver le poids des entrées et la coupe de l'axe qui convient pour représenter ces données ?

La réponse est oui, mais pas facilement. Si nous essayons de deviner le poids des entrées et la coupe de l'axe au hasard, nous obtiendrons des résultats différents des résultats cibles. Par exemple, si nous supposons que le poids des entrées est de 2 et que la coupe de l'axe est de 0, notre fonction sera :

$$°F = (°C \times 1) + 0$$

Cette fonction donne des résultats imprécis. Par exemple, si nous voulons convertir $20°C$ en Fahrenheit, elle donne : [7]

$$°F = (20 \times 1) + 0 = 20$$

Et c'est une erreur car la valeur correcte est $68°F$.

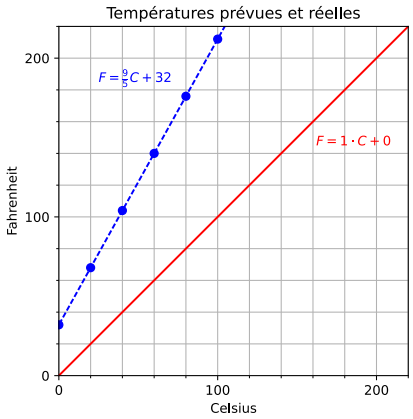

Figure 6: L'écart entre les températures prévues et réelles

Alors, comment trouvons-nous le poids des entrées et la coupe de l'axe corrects ? C'est là que l'apprentissage en profondeur intervient. L'apprentissage en profondeur utilise un algorithme appelé régression pour trouver les meilleures valeurs pour ces deux facteurs afin de réduire l'écart entre les résultats générés et les résultats cibles. Cet écart est appelé fonction de coût ou fonction d'erreur.

I.5. Erreur quadratique moyenne MSE,

La fonction de coût est une fonction mathématique mesurée entre zéro et la valeur maximale possible. Plus la valeur de la fonction de coût est proche de zéro, plus les résultats générés sont proches des résultats cibles. Par exemple, nous utilisons la fonction de coût appelée erreur quadratique moyenne MSE, qui calcule la moyenne de toutes les mesures d'erreur quadratique entre chaque résultat généré et chaque résultat cible.

$$\text{MSE} = \frac{1}{n} \cdot \sum (y - y_0)^2$$

Où n est le nombre de paires d'entrées et de résultats cibles, y (ou $°F$) est la mesure du résultat généré et y_0 (ou $°F_0$) est la mesure du résultat cible.

Ainsi, si nous utilisons la MSE pour mesurer l'écart entre une fonction linéaire et un tableau de conversion des températures Celsius en Fahrenheit, la valeur de la MSE sera :

$$\begin{aligned}\text{MSE} &= \frac{1}{n} \cdot \sum (y - y_0)^2 \\ &= \frac{1}{n} \cdot \sum ((wx + b) - y_0)^2 \\ &= 4480\end{aligned}$$

Il s'agit d'une explication de l'algorithme de descente de gradient[5] qui est utilisé pour trouver les meilleurs poids d'entrée et les biais afin que la valeur de la fonction de coût soit réduite à zéro. Cela se fait en commençant par des poids et des biais aléatoires, puis en les mettant à jour fréquemment en se déplaçant dans la direction opposée du gradient de la fonction de coût.

I.6. Algorithme de descente de gradient

Le gradient est un vecteur qui indique la direction dans laquelle la fonction de coût augmente. En se déplaçant dans la direction opposée, nous pouvons trouver le point le plus bas de la fonction de coût, qui correspond aux meilleures valeurs pour les poids et les biais. La règle de mise à jour pour les poids et les biais est donnée par:

$$w_{n+1} = w_n - \alpha * \frac{\partial \text{MSE}}{\partial w}$$

$$b_{n+1} = b_n - \alpha * \frac{\partial \text{MSE}}{\partial b}$$

Où α est appelé taux d'apprentissage et est un petit nombre positif qui contrôle la taille du pas que nous prenons à chaque itération pour réduire la différence entre les résultats attendus et initiaux. $\frac{\partial \text{MSE}}{\partial w}$ et $\frac{\partial \text{MSE}}{\partial b}$ sont les dérivées partielles de la fonction de coût par rapport aux poids et aux biais respectivement. Ces dérivées partielles nous disent dans quelle mesure la fonction de coût change lorsque le poids ou le biais change légèrement.

Pour calculer ces dérivées partielles, nous pouvons utiliser une technique appelée règle de chaîne[8], qui nous permet de décomposer une fonction complexe en fonctions plus simples et de multiplier leurs dérivées. Par exemple, si nous avons une fonction $f(x) = g(h(x))$, où g et h sont des fonctions plus simples, nous pouvons écrire:

$$\frac{\partial f}{\partial x} = \frac{\partial g}{\partial h} * \frac{\partial h}{\partial x}$$

En utilisant cette technique, nous pouvons trouver les dérivées partielles de MSE par rapport à w et b comme suit:

$$\frac{\partial \text{MSE}}{\partial w} = \frac{1}{n}\sum(-2x(y - y_0))$$
$$\frac{\partial \text{MSE}}{\partial b} = \frac{1}{n}(-2(y - y_0))$$

Et finalemet:

$$w_{n+1} = w_n + \alpha \sum x(y - y_0)$$
$$b_{n+1} = b_n + \alpha \sum (y - y_0))$$

Où x est la valeur d'entrée $°C$, y_0 est la valeur cible de sortie $°F$ et y est la valeur de sortie obtenue en utilisant notre fonction linéaire. en choisie $\alpha = 2\frac{\alpha}{n}$ car α est un nombre arbitraire comme epsilon. Cela signifie que vous pouvez simplifier les formules en éliminant le facteur $\frac{2}{n}$, ce qui ne change pas le sens de l'algorithme.

En utilisant ces formules, on peut mettre à jour notre poids et notre biais à chaque itération jusqu'à ce qu'on atteigne un point où la fonction de coût est réduite au minimum.

Et on peut programmer un code simple en language C qui effectue cette tâche.

```c
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
// Train Data
float td[][2] = {
    // C   F
    {-40, -40},
    {-20, -4 },
    {0,   32 },
    {20,  68 },
    {40,  104},
    {60,  140},
    {80,  176},
    {100, 212},
};
#define N 8 // Number of item in Train Data
#define ALPHA 0.00001 // Define the learning alpha
#define EPOCS 100 * 1000 // Define the number of iterations
```

```
// Define a function to compute the mean squared error
double cost(double w, double b) {
  double error = 0.0;
  for (int i = 0; i < N; ++i) {
    double x = td[i][0];
    double y = td[i][1];
    double d = y - (w * x + b);
    error += d * d;
  }
  return error / (int) N;
}

// Define a function to perform gradient descent
void gradient_descent(double *w, double *b) {
  // Derivative of cost function with respect to w or b
  double dw = 0.0;
  double db = 0.0;
  for (int i = 0; i < N; i++) {
    double x  = td[i][0];
    double y0 = td[i][1];
    double y  = *w * x + *b;
    dw += x * (y - y0);
    db += (y - y0);
  }
  // Update w and b using the learning rate and the derivatives
  *w = *w - ALPHA * dw;
  *b = *b - ALPHA * db;
}

// Define a function to train the neuron using gradient descent
void train(double *w, double *b) {
  for (int i = 0; i < EPOCS; i++) {
    gradient_descent(w, b);
    if (i % 101000 == 0)
        printf("Iteration: %d, Cost:%3.3f w=%.6lf b=%.6lf\n", i,
cost(*w, *b), *w, *b);
  }
}

// Define a function to predict the output using the neuron
double predict(double x, double w, double b) {
  return w * x + b;
}
```

```c
int main() {
  // Initialize w and b randomly
  double w = (double) rand() / RAND_MAX;
  double b = (double) rand() / RAND_MAX;

  // Train the neuron using gradient descent
  train(&w, &b);

  // Print the final values of w and b
  printf("\nFinal values are: w = %.6f and b = %.6f\n\n", w, b);

  // Test the neuron with some new inputs
  double x_new = 50;                    // Celsius
  double y_new = predict(x_new, w, b); // Fahrenheit
  printf("Fahrenheit of 50C: 122F\n");
  printf("Prediction of 50C: %.6fF\n", y_new);

  return 0;
}
```

```
Iteration:  10000, Cost:227.982250 w=1.980511 b=13.953147
Iteration:  20000, Cost: 74.392002 w=1.903113 b=21.691058
Iteration:  30000, Cost: 24.274565 w=1.858902 b=26.111201
Iteration:  40000, Cost:  7.920939 w=1.833647 b=28.636129
Iteration:  50000, Cost:  2.584651 w=1.819220 b=30.078449
Iteration:  60000, Cost:  0.843387 w=1.810979 b=30.902348
Iteration:  70000, Cost:  0.275202 w=1.806272 b=31.372986
Iteration:  80000, Cost:  0.089800 w=1.803583 b=31.641830
Iteration:  90000, Cost:  0.029302 w=1.802046 b=31.795402
Iteration: 100000, Cost:  0.009562 w=1.801169 b=31.883127

Final values are: w = 1.801169 and b = 31.883127

Fahrenheit of 50C: 122F
Prediction of 50C: 121.94F
```

I.7. Dicsussion

Dans l'entraîner de ce modèle, on a utilisé un ensemble de données d'entraînement (td) qui contient des paires de températures en Celsius et en Fahrenheit, et qui cherche à apprendre la formule de conversion entre ces deux unités. La formule exacte est $y = \frac{9}{5}x + 32$, où y est la température en Fahrenheit et x est la température en Celsius. On a initialisé les paramètres du modèle à des valeurs

aléatoires proches de zéro, et on a lancé la descente de gradient pour 100000 itérations. A chaque 10000 itérations, on a affiché l'évolution de la fonction de coût et des paramètres du modèle.

Après 100000 itérations, on a obtenu les résultats suivants :

- Fonction de coût : 0.009562
- Poids : 1.801169
- Biais : 31.883127

On peut voir que les paramètres du modèle sont très proches des valeurs exactes de la formule de conversion. Pour tester la performance du modèle, on a utilisé une nouvelle température $50C$ qui n'est l'entrainner pas et on obient une temperature de $121.94F$, On a calculé l'erreur moyenne absolue (MSE) entre les prédictions du modèle et les valeurs réelles. On a obtenu un MSE de $0.029°C$, ce qui montre que le modèle est très précis et qu'il a bien appris la formule de conversion.

On peut visualiser les résultats du modèle sur un graphique qui montre la relation entre les températures en Celsius et en Fahrenheit. On peut voir que les points sont alignés sur une droite qui correspond à la formule y = 1.8x + 32. On peut également comparer le modèle avec un modèle aléatoire qui prédit des valeurs aléatoires entre -40℃ et 100℃. On peut voir que le modèle aléatoire a un MAE beaucoup plus élevé que le modèle entraîné.

On peut conclure que le modèle de régression linéaire[9] [10] est capable de générer des prédictions très proches des valeurs réelles, et qu'il a réussi à apprendre la formule de conversion entre les températures en Celsius et en Fahrenheit. Ce modèle pourrait être utilisé pour convertir des températures dans d'autres unités, comme les kelvins ou les degrés Rankine.

CONCLUSION

Dans ce chapitre, on a vu ce qu'est le deep learning, comment il fonctionne. On a appris que le deep learning est une branche de machine learning qui utilise des réseaux de neurones artificiels multicouches pour apprendre à partir de données complexes ou de haute dimension. On a vu que le deep learning se distingue de l'apprentissage traditionnel par le fait qu'il ne repose pas sur des règles ou des algorithmes prédéfinis, mais qu'il peut générer ses propres règles et algorithmes par essai et erreur[5]. On a aussi vu que le deep learning peut surmonter certains des problèmes rencontrés par l'apprentissage traditionnel, tels que le bruit, le manque ou le changement des données.

On a également compris le principe de l'apprentissage profond en utilisant des exemples de notre vie quotidienne. On a vu comment on peut utiliser une fonction mathématique appelée fonction coût pour mesurer la différence entre les résultats d'un modèle d'apprentissage et les résultats souhaités ou corrects. Puis on a vu comment on peut utiliser une autre fonction appelée fonction optimisation pour ajuster la valeur de chaque cellule neuronale dans le réseau d'apprentissage afin de réduire la valeur de la fonction coût. Ces étapes sont répétées sur un grand ensemble de données jusqu'à ce que le modèle d'apprentissage soit capable d'accomplir les tâches demandées avec précision ou acceptabilité.

On a aussi exploré les réseaux de neurones artificiels, qui sont des modèles d'intelligence artificielle qui utilisent des cellules nerveuses artificielles pour convertir les entrées en sorties[1]. On a vu comment chaque cellule nerveuse reçoit des signaux d'autres cellules et envoie des signaux à d'autres cellules. Chaque signal est ajouté à une valeur de poids qui détermine sa force et son importance. Chaque cellule nerveuse calcule la somme des signaux reçus et applique une fonction d'activation pour produire un signal de sortie.

Enfin, on a donné un exemple de deep learning dans la pratique en utilisant un code simple en langage C qui effectue une régression linéaire pour convertir la température de Celsius en Fahrenheit. On a vu comment on peut collecter un ensemble de données qui forment des paires d'entrées et de résultats cibles, comment on peut initialiser les paramètres du modèle à des valeurs aléatoires proches de zéro, comment on peut lancer la descente de gradient pour mettre à jour les paramètres du modèle à chaque itération, et comment on peut tester

la performance du modèle sur une nouvelle température qui n'est pas dans l'ensemble d'entraînement.

On peut conclure que le deep learning est un domaine récent et prometteur en informatique qui mérite l'attention et la recherche, et qui peut contribuer à résoudre de nombreux problèmes dans différents domaines tels que la traduction, la reconnaissance d'images et de sons, etc. Le deep learning permet aux systèmes de regrouper les données et de faire des prédictions avec une précision incroyable. Le deep learning s'inspire de la structure du cerveau humain et tente de tirer des conclusions similaires à celles que les humains feraient en analysant continuellement les données avec une structure logique donnée. Le deep learning utilise des structures multicouches d'algorithmes appelées réseaux neuronaux, qui peuvent extraire les sens et les motifs cachés dans les données sans avoir besoin d'intervention humaine[1]■.

RÉFÉRENCES BIBLIOGRAPHIQUES.

[1] Wikipédia, "Apprentissage profond," Wikimedia Foundation, 2021. [Online]. Available: https://fr.wikipedia.org/wiki/Apprentissage_profond

[2] "What is deep learning?," . [Online]. Available: https://www.ibm.com/topics/deep-learning (IBM)

[3] J. Brownlee, "9 applications of deep learning for computer vision," *Mach. Learn. Mastery*, 2019. [Online]. Available: https://machinelearningmastery.com/applications-of-deep-learning-for-computer-vision/

[4] S. S. Zumdahl, and S. A. Zumdahl, *Chemistry*, Cengage Learning, 2013.

[5] I. Goodfellow, Y. Bengio, and A. Courville, "Deep learning," *MIT Press*, 2016.

[6] D. P. Kingma, and J. Ba, "Adam: a method for stochastic optimization," *Arxiv Preprint Arxiv:1412.6980*, 2014.

[7] S. R. Dubey, S. K. Singh, and B. B. Chaudhuri, "A comprehensive survey and performance analysis of activation functions in deep learning," *Corr*, 2021. [Online]. Available: https://arxiv.org/abs/2109.14545

[8] M. Spivak, *Calculus*, Cambridge University Press, 1967.

[9] Wikipedia, "Linear regression," *Wikipedia*, 2021. [Online]. Available: https://en.wikipedia.org/wiki/Linear_regression

[10] Wikipedia, "Régression linéaire," *Wikipedia*, 2021. [Online]. Available: https://fr.wikipedia.org/wiki/R%C3%A9gression_lin%C3%A9aire

Chapitre I.

DETECTING LUNG CANCER NODULES

I.1. introduction

The project is to create a detector for lung cancer, and based on the **LUNA dataset** luna16.grand-challenge.org that is a collection of CT scans of patients with lung nodules, which are small growths in the lungs that may indicate cancer. The dataset is part of a Grand Challenge, which is a competition among researchers to develop and test methods for detecting and classifying nodules. The dataset is open and publicly available,

LUNA16 It contains *1,186 lung nodules* annotated in *888 CT scans* by 4 experienced radiologists. The LUNA dataset has two tracks: one for finding the locations of nodules in the scans, and another for reducing false positives by distinguishing benign from malignant nodules.

Automating this process will provide an experience in dealing with difficult scenarios where solving problems is challenging. Automatic detection of lung cancer is challenging, and even professional specialists face difficulty in identifying malignant tumors. Automating the process with deep learning will be more demanding and require a structured approach to succeeding.

Detecting lung cancer early is essential for increasing the patient's survival rate, but it's tough to do manually, especially on a large scale. The problem space of lung tumor detection is important because it is an active research area with promising results. However, it is also unsolved, which satisfies the authors' objective of using PyTorch to tackle state-of-the-art projects.

In large-scale project, it will be working with 3D data and require data manipulation, as no pre-built library is available for suitable training samples. The project will involve using convolutional layers followed by a resolution-reducing downsampling layer. To handle the computational requirements, you will need access to a GPU with at least

```
8 GB
```

of RAM or

```
220 GB
```

of free disk space for raw training data, cached data, and trained models.

Instead of analyzing the entire CT scan, it will break down the problem into simpler tasks. CT scans are 3D X-rays consisting of a three-dimensional array of single-channel data, with each voxel having a numeric value that approximately corresponds to the average mass density of the matter contained inside.

As for choosing the batch size, it depends on your specific situation. For example, with an image size of 2400x2400x3x4, a single image takes 70 MB, so a batch size of 5 might be more realistic. However, this depends on the available GPU memory, and using 16-bit values instead of 32-bit can help double the batch size

I.1.1. Definitions.

- **CNN**: The design of a convolutional neural network for detecting tumors are based on alternative image recognition that can be used as a starting point. This Convolutional neural networks typically have a tail, backbone, and head. The tail processes the input, while the backbone contains most of the layers arranged in series of blocks. The head converts the output from the backbone to the desired output form.
- **Epoch Training**: The epoch is divided into 20193 steps called batches, each containing 256 data points. Once the data is loaded and the training process begins, monitoring the performance of the computing resources is crucial to ensure that resources are being used effectively.
- **Metrics**: displays training and validation metrics in a graphical format, making it easier to interpret the data. We can adjust the smoothing option to remove noise from trend lines if our data is noisy.
  - Recall is the ability to identify all relevant things.
  - while precision is the ability to identify only relevant things.

The logging output are include the precision by including the count of correctly identified and the total number of samples for both negative and positive samples.

- **Overfitting**: Overfitting occurs when a model learns specific properties of the training set, losing the ability to generalize, and making it less accurate in predicting samples that haven't been trained on. For instance. To avoid overfitting, we must examine the right metrics. Looking at our overall loss, everything

might seem fine, but that's because our validation set is unbalanced, and the negative samples dominate, making it hard for the model to memorize individual details. To prevent overfitting, we use data augmentation, which involves modifying a dataset by applying synthetic alterations to individual samples, resulting in a new dataset with a larger number of effective samples. Five specific data augmentation techniques are discussed, including mirroring the image, shifting it by a few voxels, scaling it up or down, rotating it around the head-foot axis, and adding noise to the image.

- **Data Automating**: This technique are designed to create new training samples from the existing ones by applying simple transformations. The transformations include shifting/mirroring, scaling, rotation, and adding noise.
- **Thresholding** is a simple and common method of segmentation that works by selecting a pixel value (called a threshold) that separates the foreground (the region of interest) from the background (the rest of the image)[^3^]. For example, if you want to segment the bone from a CT scan, you can choose a threshold that corresponds to the intensity of bone pixels and ignore the pixels that are lower or higher than that value. However, thresholding is not always accurate or robust, especially when dealing with complex or noisy images.

I.1.2. Approach for Training our Model involves five main steps

The goal of this project is to create an end-to-end solution for detecting cancerous nodules in lung CT scans using PyTorch. The approach involves five main steps:

1. Loading the CT data and converting it into a PyTorch dataset.
2. Segmenting the image to identify potential tumors.
3. Grouping interesting voxels to form candidates.
4. Classifying the nodules using a classification model.
5. Diagnosing the patient based on the malignancy of the identified nodules, combining segmentation and classification models for a final diagnosis.

I.2. step 1: Manipulation the Data

I.2.1. Data Conversions

To process the data, it is necessary to convert raw data files into a format that is usable by PyTorch, which means converting the row data from 3D array of intensity data to

pyTorch format. This data is around 32 million voxels, which is much larger than the nodules. To make the task more manageable, the model will focus on a relevant crop of the CT scan. There are various steps involved in processing the data, including understanding the data, mapping location information to array indexes, and converting the CT scan intensity into mass density. Identifying the key concepts of the project, such as nodules, is crucial.

I.2.2. Data loading

The first step in creating a neural network for detecting lung cancer using PyTorch is handling the dataset. The goal is to produce a training sample from raw CT scan data and a list of annotations, by following this topics:

- Loading and processing raw data files
- Implementing a Python class to represent the data
- Converting the data into a format usable by PyTorch
- Visualizing the training and validation data

Overall, the quality of the data used to train the model has a significant impact on the project's success.

I.2.3. Raw CT Data Files

Loading CT data and processes the information to produce a 3D array, and transforms the patient coordinate system to the index, row, and column coordinates of each voxel in the array.

Annotation data from LUNA with nodule coordinates and malignancy flags are also loaded. It contains information about all lumps that look like nodules, whether they are malignant, benign, or something else. We'll use this to build a list of candidates that can be split into training and validation datasets. This Dataset contains information about some of the candidates that have been flagged as nodules, including the diameter. This information is useful for ensuring a representative range of nodule sizes in the training and validation data.

I.2.4. Training and validation sets

Splitting a dataset into training, validation, and test sets is a crucial step in building a machine learning model. It allows for the model to be trained on one set, tuned on another, and evaluated on a final set. This helps prevent overfitting and gives an accurate measure of the model's performance.

We want to ensure that both sets represent the real-world input data that we expect to see and handle normally. If either set is significantly different from our actual use cases, it's highly likely that our model will behave differently than we expect. This split helps us evaluate and improve the model's performance before we deploy it on production data.

I.2.5. Loading individual CT scans

We need to understand how to load and understand CT scan data, which is usually stored in a DICOM file format. The MetaIO format is suggested for easier use, and the Python SimpleITK library can be used to convert it to a NumPy array. The Hounsfield Unit (HU) scale is used to measure CT scan voxel density, with air at -1000 HU, water at 0 HU, and bone at least 1000 HU.

I.2.6. Data Ranges and Model Inputs

Starting with adding channels of information to our samples. To prevent the overshadowing of the new channels by raw HU values, we must be aware that our data ranges from -1,000 to +1,000. We won't add more channels of data for the classification step, so our data handling will remain the same.

Fixed-size inputs are necessary due to a fixed number of input neurons. We want to train our model using a crop of the CT scan that accurately centers the candidate, making identification easier for the model by decreasing the variation in expected inputs.

I.2.7. The Patient Coordinate System

The candidate center data expressed in millimeters, not voxels. We need to convert our coordinates from the millimeter-based coordinate system (X, Y, Z) to the voxel-address-based coordinate system (I, R, C). The patient coordinate system defines the positive X to be patient left, positive Y to be patient behind, and the positive Z to be toward patient head. The patient coordinate system is often used to specify the locations of interesting anatomy in a way that is independent of any particular scan.

I.2.8. CT Scan Shape and Voxel Sizes

The size of the voxels varies between CT scans and typically are not cubes. The row and column dimensions usually have voxel sizes that are equal, (...) and the index dimension has a larger value, but other ratios can exist.

I.2.9. Converting Between Millimeters and Voxel Addresses

(...) Converting between patient coordinates in millimeters and (I, R, C) array coordinates, we define some utility code to assist with the conversion. Flipping the axes is encoded in a 3×3 matrix. The metadata we need to convert from patient coordinates to array coordinates is contained in the MetaIO file alongside the CT data itself.

(...) In CT scan images of patients with lung nodules, most of the data is not relevant to the nodule (up to 99.9999%). To extract the nods, an area around each candidate will be extracted, so the model can focus on one candidate at a time.

I.3. step 2: Segmentation

The process of segmentation to identify possible nodules, which is step 2 of the project's plan. The segmentation model is created using a **U-Net**. The objective is to flag voxels that might be part of a nodule and use the classification step to reduce the number of incorrectly marked voxels.

I.3.1. Semantic segmentation: Per-pixel classification

U-Net network is a popular architecture for semantic segmentation. It was originally proposed for biomedical image segmentation, but since then it has been widely used for several other domains such as self-driving cars, satellite imagery, and more. The U-Net architecture includes an encoder that down-samples the input image, which is then followed by an up-sampling decoder. This allows the network to learn high-level semantic features while preserving the spatial information, making it suitable for semantic segmentation.

Semantic segmentation identifies different objects and where they are in a given image. If there are multiple cats in an image, semantic segmentation can identify each cat's position. The existing classification models can't pinpoint where the cat is; they can only predict whether or not a cat is present in the image.

Semantic segmentation requires combining raw pixels to develop specific detectors for items like color and then building on this to create more informative feature detectors to finally identify specific things like a cat or a dog. Nonetheless, the segmentation model will not give us a single classification-like list of binary flags like classification models since the output should be a heatmap or mask.

I.3.2. Why we need heatmap or mask as output of segmentation

The output of a U-Net network in biomedical image segmentation is typically a heatmap or a mask because these formats provide a clear visual representation of the boundaries that the network has identified in the image. A heatmap is a colored image that highlights the regions of the input image that are most important for the output classes, whereas a mask is a binary image that indicates which pixels belong to which class.

In biomedical image segmentation, it is important to accurately identify the areas of interest, such as tumors or blood vessels, to aid in diagnosis and treatment. The heatmap or mask allows for easy visualization of these areas and can be used by medical professionals to make more informed decisions. Moreover, the heatmap or mask output can be used as input for further processing and analysis, such as quantifying the size or volume of a segmented region.

I.3.3. UNet Architecture for Image Segmentation

U-Net is a convolutional neural network that can produce pixelwise output for image segmentation. It has a U-shaped encoder-decoder structure that operates at different resolutions. The encoder network reduces the spatial dimensions and increases the number of filters at each block, while the decoder network does the opposite. The key innovation of U-Net is the use of skip connections that link the encoder and decoder blocks at the same level, allowing the network to capture multi-scale features and produce more precise segmentations.

We use the same source data as before: CT scans and annotation data. Our goal is to create bounding boxes that cover the whole nodules based on their annotated centers. This will help us with segmentation, which is the process of identifying regions of interest in images. To do this, we search for voxels with high density around the center of each nodule on the row and column axis. We stop when we reach voxels with low density, which indicate normal lung tissue. Then we repeat the search in the third dimension.

We have seven input channels for UNet: three context slices before and after the focus slice, and one focus slice that we segment. We have one output class indicating whether a voxel is part of a nodule.

I.3.4. Visualizing CT Image with Predicted Nodules

The U-Net model has an encoder that captures the context of the image and a decoder that produces the segmentation map. To evaluate the performance of the model, we create an empty image with 512x512 pixels and three color channels.

We overlay the predicted segmentation map on the original CT image and use different colors to indicate the errors. We use red for false positives (pixels that are predicted as nodules but are not), green for true positives (pixels that are predicted as nodules and are), and orange for false negatives (pixels that are not predicted as nodules but are).

The pixel values are normalized between 0 and 1. The goal is to have a clear visualization of the nodules and the errors in the prediction.

I.4. step 3: Grouping nodules.

We use segmentation to find ROIs that might be nodules in CT images. Then we group pixels that are next to each other and above the limit. Each group is a nodule candidate with a center point (index, row, column). We use these points to classify the candidates. Grouping makes the search easier and removes noise.

I.5. step 4: Classification

This step involves dividing the CT scan into individual slices. The output of the segmentation step is an array of per-pixel probabilities, indicating whether the pixel is part of a nodule. These slice-wise predictions are collected in a mask array with the same shape as the CT scan input, and a threshold is applied to the predictions to obtain a binary array. A cleanup step which shortens the flagged area and removes small components.

I.6. step 5: Diagnosing the patient

We use the LIDC annotations to decide if a nodule (a small lump) in the lung is cancerous or not. The LIDC annotations are labels that up to four doctors gave to each nodule based on how it looks in a CT scan. They used a scale from 1 to 5, where 1 means the nodule is very unlikely to be cancerous and 5 means it is very likely to be cancerous. These labels are not based on other information about the patient, such as their medical history or symptoms. To make a final decision, we will use a rule that says a nodule is cancerous if at least two doctors gave it a 4 or a 5. This rule is not very precise and there are other ways of using the labels, such as taking the average or ignoring some nodules.

I.7. Conclusion

The identifying nodule candidates in CT scans for possible cancer detection. A connected-components algorithm is used for grouping the suspected nodule voxels. The labeled chunks are passed on to a classification module to reduce false positives. Finally, the identified regions in the CT scan are cropped and passed onto the classification module using DataLoader.

We use a data loader, to loop over a candidate list to threshold. The output probabilities to get a list of things our model thinks are actual nodules, which would be output for a radiologist to inspect while adjusting the threshold to err a bit on the safe side. A single CT scan from the validation set is run, and 16 nodule candidates are found.

The task of identifying malignant nodules from benign ones in CT scans after implementing the nodule-detection task of the LUNA challenge. Even with a good system, diagnosing malignancy would need a more comprehensive view of the patient, additional non-CT context, and a biopsy instead of just looking at a particular nodule in isolation on a CT scan. This task is likely to be performed by a doctor for some time to come.

- Splitting training and validation (and test) sets between patients is important to avoid errors.
- Converting pixel-wise marks to nodules can be achieved using traditional image processing.
- The diagnosis performs both segmentation and classification.
- TensorBoard can help us visualize and identify network anomalies.
- There is no magic bullet when training neural networks.

This system is not ready to replace a human radiologist, but it could be a useful tool to help them find suspicious areas in the scans. And would need more data and validation from experts, as well as regulatory approval from authorities. The system would also need to run in a scalable environment that can handle different cases and situations.