جامعة عبد المالك السعدي
ⵜⴰⵙⴷⴰⵡⵉⵜ ⵏ ⵄⴰⴱⴷⵍⵎⴰⵍⵉⴽ ⵙⵙⴰⵄⴷⵉ
Université Abdelmalek Essaâdi

المدرسة الوطنية للعلوم التطبيقية بتطوان
ⵜⵉⵎⵙⵙⵓⵔⵉⵏ ⵜⵏⴰⴹⵕⵉⵜ ⵏ ⵜⵙⴽⵔⴰⵙⵉⵏ ⵜⵉⵜⵡⵉⵏⵉⵏ
ÉCOLE NATIONALE DES SCIENCES APPLIQUÉES DE TÉTOUAN

Prédiction du Marché Boursier

Réalisé par :

Hiba Kharfasse

Hafsa Ounad

Hamza Marir

Encadré par :

MR. Anas Belcaid

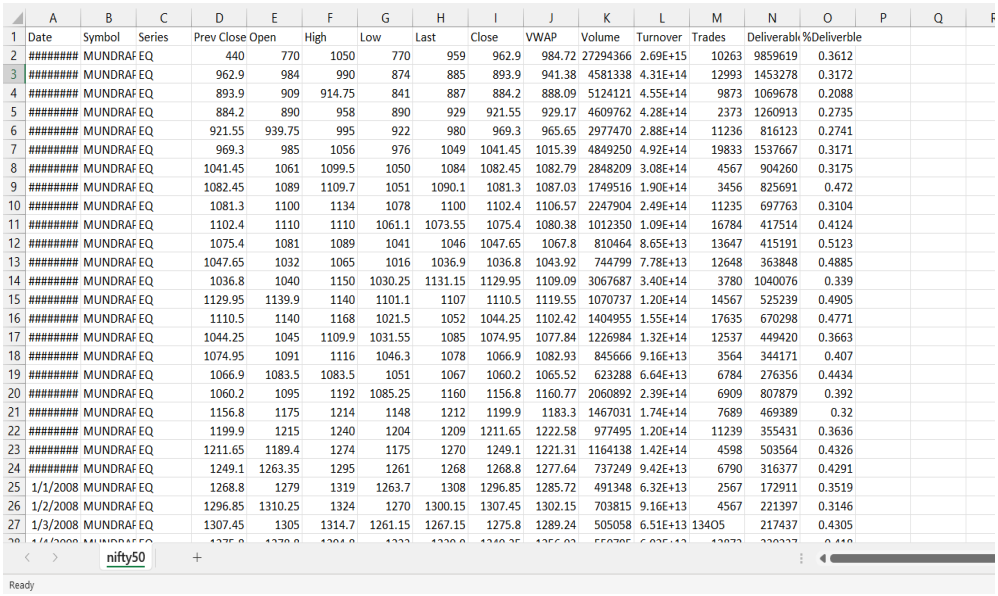
Remerciements

Ce projet n'aurait pas pu voir le jour sans le soutien et l'implication de nombreuses personnes à qui nous souhaitons adresser nos sincères remerciements.
Nous exprimons notre profonde gratitude à l'école nationale des sciences appliquées de Tetouan pour avoir offert un environnement favorable à la réalisation de ce projet et pour les ressources mises à notre disposition.
Un immense merci à notre prof, MR. Anas Belcaid pour ses conseils éclairés, sa disponibilité et sa patience tout au long de cette expérience. Ses retours constructifs et son expertise ont été essentiels à l'aboutissement de ce travail.
Nous remercions également l'ensemble des membres de notre équipe pour leur collaboration et leurs échanges enrichissants, qui ont largement contribué à la qualité et à la pertinence de ce projet.
Enfin, nos pensées reconnaissantes vont à nos proches pour leur soutien moral et leur compréhension, qui nous ont permis d'aborder ce défi avec sérénité et motivation.
Ce projet est le résultat d'un effort collectif, et nous sommes reconnaissants envers toutes les personnes qui ont, de près ou de loin, participé à cette belle aventure.
Merci à tous.

Introduction générale

Dans un monde de plus en plus interconnecté, les marchés financiers jouent un rôle central dans l'économie mondiale. La prévision des fluctuations du marché boursier est une tâche complexe en raison de la multitude de facteurs influents, notamment les tendances économiques, les actualités politiques, et les comportements des investisseurs. Cette complexité a conduit à l'émergence de solutions basées sur des approches avancées en intelligence artificielle et en apprentissage profond (deep learning).
Parmi ces approches, les réseaux de neurones convolutifs (CNN) se distinguent par leur capacité à extraire automatiquement des caractéristiques pertinentes à partir de données complexes, notamment des représentations graphiques ou des matrices de densité issues des données boursières. Ces modèles, initialement conçus pour le traitement des images, s'avèrent également efficaces pour analyser les relations temporelles et structurelles des données financières.
Ce projet s'inscrit dans cette dynamique en explorant deux approches complementaires : l'utilisation de modèles CNN pré-entraînés et le développement d'un modèle CNN construit "from scratch". L'objectif est d'évaluer leur efficacité respective pour prédire les variations des marchés financiers, notamment les indicateurs essentiels tels que les prix d'ouverture (Open), de clôture (Close), le plus haut (High) et le plus bas (Low) des actions boursières.
Ce rapport détaille les étapes clés du projet, à commencer par la préparation des données et la génération de représentations adaptées aux CNN, jusqu'à la construction et l'évaluation des modèles. À travers cette étude, nous cherchons à identifier les forces et les limites de chaque approche, tout en contribuant à l'amélioration des outils de prévision boursière basés sur le deep learning.

Dataset



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	Date	Symbol	Series	Prev Close	Open	High	Low	Last	Close	VWAP	Volume	Turnover	Trades	Deliverabl	%Deliverble
2	########	MUNDRAF	EQ	440	770	1050	770	959	962.9	984.72	27294366	2.69E+15	10263	9859619	0.3612
3	########	MUNDRAF	EQ	962.9	984	990	874	885	893.9	941.38	4581338	4.31E+14	12993	1453278	0.3172
4	########	MUNDRAF	EQ	893.9	909	914.75	841	887	884.2	888.09	5124121	4.55E+14	9873	1069678	0.2088
5	########	MUNDRAF	EQ	884.2	890	958	890	929	921.55	929.17	4609762	4.28E+14	2373	1260913	0.2735
6	########	MUNDRAF	EQ	921.55	939.75	995	922	980	969.3	965.65	2977470	2.88E+14	11236	816123	0.2741
7	########	MUNDRAF	EQ	969.3	985	1056	976	1049	1041.45	1015.39	4849250	4.92E+14	19833	1537667	0.3171
8	########	MUNDRAF	EQ	1041.45	1061	1099.5	1050	1084	1082.45	1082.79	2848209	3.08E+14	4567	904260	0.3175
9	########	MUNDRAF	EQ	1082.45	1089	1109.7	1051	1090.1	1081.3	1087.03	1749516	1.90E+14	3456	825691	0.472
10	########	MUNDRAF	EQ	1081.3	1100	1134	1078	1100	1102.4	1106.57	2247904	2.49E+14	11235	697763	0.3104
11	########	MUNDRAF	EQ	1102.4	1110	1110	1061.1	1073.55	1075.4	1080.38	1012350	1.09E+14	16784	417514	0.4124
12	########	MUNDRAF	EQ	1075.4	1081	1089	1041	1046	1047.65	1067.8	810464	8.65E+13	13647	415191	0.5123
13	########	MUNDRAF	EQ	1047.65	1032	1065	1016	1036.9	1036.8	1043.92	744799	7.78E+13	12648	363848	0.4885
14	########	MUNDRAF	EQ	1036.8	1040	1150	1030.25	1131.15	1129.95	1109.09	3067687	3.40E+14	3780	1040076	0.339
15	########	MUNDRAF	EQ	1129.95	1139.9	1140	1101.1	1107	1110.5	1119.55	1070737	1.20E+14	14567	525239	0.4905
16	########	MUNDRAF	EQ	1110.5	1140	1168	1021.5	1052	1044.25	1102.42	1404955	1.55E+14	17635	670298	0.4771
17	########	MUNDRAF	EQ	1044.25	1045	1109.9	1031.55	1085	1074.95	1077.84	1226984	1.32E+14	12537	449420	0.3663
18	########	MUNDRAF	EQ	1074.95	1091	1116	1046.3	1078	1066.9	1082.93	845666	9.16E+13	3564	344171	0.407
19	########	MUNDRAF	EQ	1066.9	1083.5	1083.5	1051	1067	1060.2	1065.52	623288	6.64E+13	6784	276356	0.4434
20	########	MUNDRAF	EQ	1060.2	1095	1192	1085.25	1160	1156.8	1160.77	2060892	2.39E+14	6909	807879	0.392
21	########	MUNDRAF	EQ	1156.8	1175	1214	1148	1212	1199.9	1183.3	1467031	1.74E+14	7689	469389	0.32
22	########	MUNDRAF	EQ	1199.9	1215	1240	1204	1209	1211.65	1222.58	977495	1.20E+14	11239	355431	0.3636
23	########	MUNDRAF	EQ	1211.65	1189.4	1274	1175	1270	1249.1	1221.31	1164138	1.42E+14	4598	503564	0.4326
24	########	MUNDRAF	EQ	1249.1	1263.35	1295	1261	1268	1268.8	1277.64	737249	9.42E+13	6790	316377	0.4291
25	1/1/2008	MUNDRAF	EQ	1268.8	1279	1319	1263.7	1308	1296.85	1285.72	491348	6.32E+13	2567	172911	0.3519
26	1/2/2008	MUNDRAF	EQ	1296.85	1310.25	1324	1270	1300.15	1307.45	1302.15	703815	9.16E+13	4567	221397	0.3146
27	1/3/2008	MUNDRAF	EQ	1307.45	1305	1314.7	1261.15	1267.15	1275.8	1289.24	505058	6.51E+13	13405	217437	0.4305

nifty50

Ready

Le dataset utilisé dans ce projet est celui du **Nifty 50**, un indice boursier qui représente les 50 plus grandes entreprises cotées à la Bourse Nationale de l'Inde (NSE) en termes de capitalisation boursière. Ce dataset contient **232 155 lignes de données**, couvrant la période du **27 novembre 2007 au 27 novembre 2020**. Il représente ainsi une large base d'informations historiques sur les actions des entreprises composant l'indice Nifty 50, ce qui permet une analyse approfondie du comportement du marché boursier indien.
Le fichier source est disponible en ligne dans KAGGLE :
https://github.com/Pranavd0828/NIFTY50-StockMarket/blob/main/NIFTY50_all.csv.zip
Le dataset est structuré en **15 colonnes**, qui contiennent les informations suivantes :

1. **Date** :Représente le jour où les données boursières ont été enregistrées.
2. **Symbol** :Le symbole boursier ou ticker qui identifie l'action.
3. **Series** :Indique le type de série de l'actif (par exemple, « EQ » pour les actions ordinaires).
4. **Prev Close** :Le prix de clôture de l'action lors de la dernière séance de trading.
5. **Open** :Le prix d'ouverture de l'action pour la séance en cours.
6. **High** :Le prix le plus élevé atteint par l'action pendant la journée.
7. **Low** :Le prix le plus bas atteint par l'action pendant la journée.

8. **Last** :Le dernier prix auquel l'action a été échangée avant la clôture du marché.

9. **Close** :Le prix de clôture officiel de l'action à la fin de la journée de trading.

10. **VWAP (Volume Weighted Average Price)** :Le prix moyen pondéré par le volume des transactions de la journée. Il montre le prix auquel la majorité des transactions ont eu lieu.

11. **Volume** :Le nombre total d'actions échangées pendant la journée.

12. **Turnover** :La valeur totale des transactions (volume multiplié par le prix).

13. **Trades** :Le nombre total de transactions effectuées sur l'action durant la journée.

14. **Deliverable Quantity** :Le nombre d'actions livrées à l'acheteur, c'est-à-dire celles qui ne sont pas revendues le même jour (pas de day trading).

15. **%Deliverable** :Le pourcentage du volume total d'actions échangées qui ont été livrées à l'acheteur.

Ce dataset fournit une base riche pour l'analyse des tendances boursières, la modélisation des comportements du marché et la prédiction des prix futurs. Chaque ligne représente les données boursières pour une journée spécifique, avec des informations essentielles sur les prix, le volume de transactions et d'autres mesures financières.

Modèles utilisés

1.CNN (from scratch) :
1.Prétraitement des Données
Avant de pouvoir utiliser le dataset Nifty 50 pour les prédictions, plusieurs étapes de prétraitement doivent être réalisées pour garantir la qualité des données. Ces étapes sont essentielles pour nettoyer et préparer les données avant de les passer à un modèle de deep learning.

1. **Gestion des Valeurs Manquantes (Missing Values)** :Le dataset peut contenir des valeurs manquantes dans certaines colonnes. Ces valeurs doivent être traitées de manière appropriée. Une approche courante consiste à remplacer les valeurs manquantes par la moyenne, la médiane ou une interpolation des valeurs adjacentes. Dans certains cas, les lignes contenant des valeurs manquantes peuvent être supprimées si elles sont trop nombreuses.

2. **Normalisation des Données** :La normalisation est cruciale pour les modèles de deep learning, car elle permet de mettre toutes les variables sur la même échelle. Nous allons normaliser les colonnes sélectionnées (Open, High, Low, Close) en utilisant une méthode de mise à l'échelle telle que la normalisation min-max ou la standardisation. Cela assurera une meilleure convergence du modèle et un apprentissage plus efficace.

3. **Sélection des Colonnes** :Pour simplifier l'analyse et la modélisation, nous allons nous concentrer uniquement sur quatre colonnes spécifiques du dataset :

   (a) **Open** : Le prix d'ouverture de l'action pour chaque journée.

   (b) **High** : Le prix le plus élevé de l'action durant la journée.

   (c) **Low** : Le prix le plus bas de l'action durant la journée.

   (d) **Close** : Le prix de clôture de l'action à la fin de la journée.

Génération des Histogrammes 2D pour la Prédiction des Valeurs
L'objectif est de prédire les valeurs futures des prix d'ouverture, des prix les plus élevés, les plus bas et des prix de clôture en utilisant des histogrammes 2D. Chaque histogramme est basé sur les valeurs de deux jours successifs. Voici un exemple détaillé de ce processus, illustré avec des tableaux.
1. Sélection des Axes pour les Histogrammes 2D
Nous allons utiliser deux jours successifs pour générer des histogrammes 2D. Le **jour n-1** est représenté sur l'axe des **X** et le **jour n** est représenté sur l'axe des **Y**. Ces axes représentent les prix d'ouverture (par exemple, "Open") des deux jours successifs.
Exemple de construction d'un histogramme 2D :

1. **Jour 1** : prix d'ouverture = 100
2. **Jour 2** : prix d'ouverture = 102
3. **Jour 3** : prix d'ouverture = 105
4. **Jour 4** : prix d'ouverture = 103
5. **Jour 5** : prix d'ouverture = 106

Chaque histogramme sera basé sur une séquence de 50 jours consécutifs, mais pour simplifier, utilisons 5 jours dans cet exemple.

Jour	Prix d'ouverture (Open)
Jour 1	100
Jour 2	102
Jour 3	105
Jour 4	103
Jour 5	106

Nous allons générer des histogrammes pour prédire le prix d'ouverture du jour 3 en utilisant les données des jours 1 et 2.
Histogramme pour la prédiction du prix d'ouverture du Jour 3 :

1. **X (Jour 1)** : 100
2. **Y (Jour 2)** : 102

L'histogramme 2D pour cette séquence est un tableau où l'axe **X** représente les prix du **Jour 1**, et l'axe **Y** représente les prix du **Jour 2**.

Jour n-1 (X)	Jour n (Y)
100	102
100	105
102	103
103	106

Chaque point dans le tableau ci-dessus représente une combinaison des valeurs de prix d'ouverture pour deux jours successifs. Ces points peuvent ensuite être organisés dans un histogramme 2D.
2. Division des Données en Bins (Catégories)
Les axes X et Y sont divisés en plusieurs **bins** ou intervalles pour mieux organiser les données. Chaque bin correspond à un intervalle spécifique de valeurs. Par exemple, si nous avons des valeurs d'ouverture comprises entre 100 et 110, nous pourrions diviser cette plage en bins de 2 points : [100, 102), [102, 104), [104, 106), [106, 108), etc.
Voici un exemple de division des données en bins pour les axes **X** et **Y** :

1. **Bins pour l'axe X** : [100, 102), [102, 104), [104, 106)
2. **Bins pour l'axe Y** : [102, 104), [104, 106), [106, 108)

Dans ce cas, pour l'exemple de la génération des histogrammes 2D, nous avons les points suivants qui se répartissent dans ces bins :

Jour n-1 (X)	Jour n (Y)	Bin X	Bin Y
100	102	[100, 102)	[102, 104)
100	105	[100, 102)	[104, 106)
102	103	[102, 104)	[102, 104)
103	106	[102, 104)	[104, 106)
106	105	[104, 106)	[104, 106)

Après avoir divisé les données en bins, l'histogramme 2D peut être construit en comptant le nombre de fois où chaque combinaison de bin apparaît dans les données.

3. Génération des Histogrammes pour les Séquences de 50 Jours

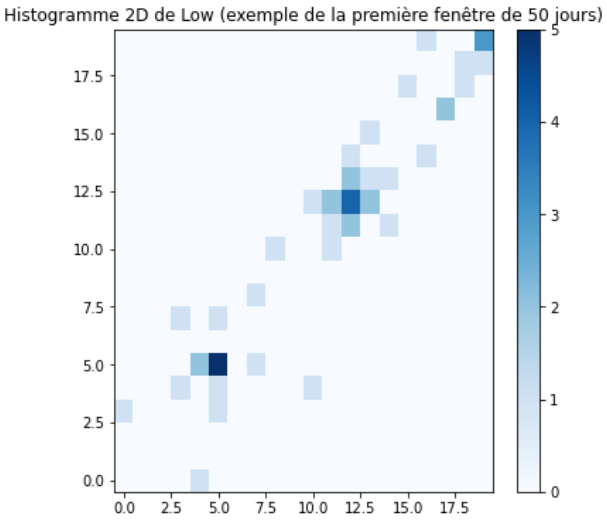
Dans cette étude, nous générons des histogrammes 2D pour des séquences de 50 jours consécutifs. Pour chaque fenêtre glissante de 50 jours, un histogramme 2D est généré. Voici comment cela se présente :

1. **Histogramme 1** : Données des jours 1 à 50
2. **Histogramme 2** : Données des jours 2 à 51
3. ...
4. **Histogramme 1000** : Données des jours 951 à 1000

Chaque histogramme est basé sur les valeurs des jours n-1 et n (par exemple, le jour 1 et 2, puis le jour 2 et 3, etc.) pour prédire le prix d'ouverture du jour suivant (Jour n).

4. Histogrammes Totaux

Dans l'étude, nous avons généré un total de **4000 histogrammes 2D** pour chaque valeur (Open, High, Low, Close). Ces histogrammes sont utilisés comme entrée pour l'entraînement des modèles de deep learning, permettant de prédire les valeurs de **Open**, **High**, **Low** et **Close** pour le jour suivant en fonction des tendances observées dans les données des jours précédents.



Les couleurs varient du blanc (faible densité) au bleu foncé (haute densité), indiquant la fréquence d'occurrence des combinaisons spécifiques de valeurs (X, Y).

2. Préparation des Données pour le Modèle

Afin de rendre les données compatibles avec le réseau neuronal convolutif, celles-ci doivent être divisées en **entrainement**, **validation** et **test**. Une séparation de 70% pour l'entraînement, 15% pour la validation et 15% pour le test est effectuée, ce qui garantit que le modèle est évalué sur un ensemble de données non vues durant l'entraînement, permettant une évaluation fiable de sa performance.

Les données sont ensuite **reshaped** (reformatées) en matrices 20x20x1. Chaque histogramme, qui est une image 2D de taille 20x20, devient ainsi un **tensor** avec une seule dimension de profondeur (**??**), correspondant à l'unique canal de couleur dans ce cas. Ces matrices seront utilisées comme entrée pour le modèle CNN.

3. Construction du Modèle CNN

1. Classe ConvLayer (Couche de Convolution)

La couche de convolution est un élément fondamental des réseaux neuronaux convolutifs (CNN). Elle est conçue pour extraire des **caractéristiques locales** à partir des images ou, dans notre cas, des histogrammes 2D. Voici le fonctionnement détaillé :

1. **Initialisation des poids et des biais** : Les poids de la couche sont initialisés aléatoirement avec une petite variance (multipliée par 0.1) pour éviter des valeurs trop grandes qui pourraient empêcher l'optimisation correcte du modèle. Les biais sont initialisés à **0**.

2. **Fonction forward (Propagation avant)** : Cette fonction calcule la sortie de la couche de convolution. Elle effectue une opération de **convolution** sur l'entrée en appliquant des filtres (ou noyaux). Chaque filtre "balaye"

l'image d'entrée et applique une opération de produit scalaire pour produire des cartes de caractéristiques (feature maps).

Ensuite, une **fonction d'activation ReLU** est appliquée pour ajouter de la non-linéarité et rendre le modèle plus expressif. ReLU remplace toutes les valeurs négatives par zéro, permettant au modèle de mieux capturer des relations complexes.

1. **Fonction backward (Rétropropagation)** : Cette fonction calcule les gradients nécessaires à la mise à jour des poids et des biais pendant l'entraînement. Elle prend en entrée le gradient de la couche suivante et le "propague" vers l'arrière, en calculant les gradients des poids, des biais et de l'entrée. Ces gradients seront utilisés par l'optimiseur pour ajuster les paramètres du modèle.

2. Classe MaxPoolLayer (Couche de Max-Pooling)
La couche de **max-pooling** est utilisée après chaque couche de convolution pour réduire les dimensions des données tout en conservant les caractéristiques les plus importantes. Elle est particulièrement utile pour **réduire la complexité du modèle** et éviter le surapprentissage (overfitting).

1. **Initialisation** : La couche de pooling est configurée avec un **taille de fenêtre de pooling** (pool_size) et un **pas de déplacement** (stride), qui déterminent la taille du sous-échantillonnage effectué.

2. **Fonction forward (Propagation avant)** : Cette fonction divise l'entrée en blocs (ou fenêtres) de la taille spécifiée par pool_size. Pour chaque bloc, elle sélectionne la valeur maximale, ce qui permet de réduire la taille des données tout en conservant l'information la plus pertinente.

3. **Fonction backward (Rétropropagation)** : Comme pour la couche de convolution, la rétropropagation est effectuée pour calculer les gradients des poids et de l'entrée. Ces gradients seront ensuite utilisés pour ajuster les paramètres du modèle lors de l'entraînement.

3. Classe DenseLayer (Couche Dense ou Fully Connected)
Une **couche dense** est une couche où chaque neurone est connecté à tous les neurones de la couche précédente. Cela permet de modéliser des relations complexes entre les caractéristiques extraites par les couches précédentes.

1. **Initialisation des poids et des biais** : Les poids de cette couche sont également initialisés de manière aléatoire avec une petite variance, et les biais sont initialisés à **0**.

2. **Fonction forward (Propagation avant)** : La sortie de cette couche est obtenue en effectuant une multiplication matricielle entre l'entrée et les poids, puis en ajoutant les biais. Cette opération est ensuite suivie par une fonction d'activation, comme **ReLU** pour les couches cachées ou **linéaire** pour la couche de sortie.

- Multiplie les entrées par les poids et ajoute les biais :
$$\text{output} = X \cdot W + b$$
- X : Entrées ($batch\_size \times input\_size$).
- W : Poids ($input\_size \times output\_size$).
- b : Biais ($1 \times output\_size$).

$$\text{ReLU}(z) = \max(0, z)$$

1. **Fonction backward (Rétropropagation)** : Lors de la rétropropagation, cette fonction calcule les gradients des poids, des biais et de l'entrée. Le gradient des poids est calculé en multipliant l'entrée transposée par le gradient de la sortie, tandis que les gradients des biais sont simplement la somme des gradients de la sortie. Le gradient de l'entrée est ensuite calculé en effectuant une multiplication matricielle avec les poids.

- Utilise la règle de dérivation pour calculer :
$$\nabla W = X^T \cdot \nabla L$$
- ∇L : Gradient de la perte par rapport à la sortie de la couche ($batch\_size \times output\_size$).
- X^T : Transposée des entrées ($input\_size \times batch\_size$).
- ∇W : Gradient des poids ($input\_size \times output\_size$).

- Le gradient des biais est calculé en sommant les gradients de sortie sur l'ensemble du batch :
$$\nabla b = \sum_{i=1}^{batch\_size} \nabla L_i$$

4. Classe AdamOptimizer (Optimiseur Adam)
L'optimiseur **Adam** (Adaptive Moment Estimation) est utilisé pour ajuster les poids du modèle pendant l'entraînement en fonction des gradients calculés. Adam est populaire car il combine les avantages de deux autres optimisateurs : **Momentum** et **RMSProp**, permettant une mise à jour des paramètres plus rapide et plus stable.

1. **Initialisation** : L'optimiseur Adam nécessite une **taux d'apprentissage** (learning_rate) pour déterminer l'ampleur des mises à jour des paramètres. De plus, deux variables sont utilisées pour chaque paramètre : **m** (momentum) et **v** (variance), qui aident à ajuster les mises à jour en fonction de l'historique des gradients.

2. **Fonction update (Mise à jour des paramètres)** : Lors de chaque étape d'entraînement, l'optimiseur effectue les étapes suivantes pour chaque paramètre (poids et biais) du modèle :

(a) **Calcul des moments** : Les moments sont calculés pour chaque paramètre en utilisant une moyenne mobile des gradients (m) et des carrés des gradients (v).

(b) **Correction des moments** : Comme le calcul des moments commence à partir de zéro, une correction est appliquée pour compenser cette initialisation.

(c) **Mise à jour des paramètres** : Les paramètres sont mis à jour en fonction des moments corrigés. La mise à jour est effectuée en utilisant le taux d'apprentissage, les moments et une petite constante pour éviter la division par zéro (1e-8).

L'optimiseur Adam ajuste efficacement les paramètres du modèle pour minimiser la fonction de perte, ce qui permet au modèle de converger rapidement vers des solutions optimales.
Voici l'architecture détaillée du modèle CNN construit dans notre code, avec les paramètres associés :
1. Couche de Convolution 1

1. **Entrée** : (batch_size, 20, 20, 1) (images 20x20, 1 canal)
2. **Paramètres** :
   (a) Nombre de filtres : 32
   (b) Taille du noyau : 3x3
   (c) Stride : 1
   (d) Activation : ReLU
3. **Sortie** : (batch_size, 18, 18, 32) (calcul : (20 - 3) + 1 = 18)

2. Max Pooling 1

1. **Paramètres** :
   (a) Taille du pooling : 2x2
   (b) Stride : 2
2. **Sortie** : (batch_size, 9, 9, 32) (calcul : (18 - 2) // 2 + 1 = 9)

3. Couche de Convolution 2

1. **Entrée** : (batch_size, 9, 9, 32)
2. **Paramètres** :
   (a) Nombre de filtres : 64
   (b) Taille du noyau : 3x3

(c) Stride : 1

(d) Activation : ReLU

3. **Sortie** : (batch_size, 7, 7, 64) (calcul : (9 - 3) + 1 = 7)

4. Max Pooling 2

1. **Paramètres** :

   (a) Taille du pooling : 2x2

   (b) Stride : 2

2. **Sortie** : (batch_size, 3, 3, 64) (calcul : (7 - 2) // 2 + 1 = 3)

5. Couche de Convolution 3

1. **Entrée** : (batch_size, 3, 3, 64)

2. **Paramètres** :

   (a) Nombre de filtres : 128

   (b) Taille du noyau : 3x3

   (c) Stride : 1

   (d) Activation : ReLU

3. **Sortie** : (batch_size, 1, 1, 128) (calcul : (3 - 3) + 1 = 1)

6. Couches Fully Connected

1. **Flatten** : La sortie de la dernière couche convolutive est aplatie en un vecteur (batch_size, 128).

2. **Fully Connected 1** :

   (a) Entrée : 128

   (b) Sortie : 64

   (c) Activation : ReLU

3. **Fully Connected 2** :

   (a) Entrée : 64

   (b) Sortie : 1

   (c) Activation : Linéaire (prédiction de valeurs continues)

7. Optimisation

1. Optimisateur : Adam

2. Learning rate : 0.001
3. Fonction de perte : Erreur quadratique moyenne (MSE)

Calculs intermédiaires de forme

1. Après **conv1** : (batch_size, 18, 18, 32)
2. Après **pool1** : (batch_size, 9, 9, 32)
3. Après **conv2** : (batch_size, 7, 7, 64)
4. Après **pool2** : (batch_size, 3, 3, 64)
5. Après **conv3** : (batch_size, 1, 1, 128)
6. Après **flatten** : (batch_size, 128)
7. Après **fc1** : (batch_size, 64)
8. Après **fc2** : (batch_size, 1)

Répartition des paramètres

1. **Couche de Convolution 1** :
   (a) Poids : 32 * (1 * 3 * 3) = 288
   (b) Biais : 32
   (c) Total : 320
2. **Couche de Convolution 2** :
   (a) Poids : 64 * (32 * 3 * 3) = 18432
   (b) Biais : 64
   (c) Total : 18496
3. **Couche de Convolution 3** :
   (a) Poids : 128 * (64 * 3 * 3) = 73728
   (b) Biais : 128
   (c) Total : 73856
4. **Fully Connected 1** :
   (a) Poids : 128 * 64 = 8192
   (b) Biais : 64
   (c) Total : 8256
5. **Fully Connected 2** :

(a) Poids : 64 * 1 = 64

(b) Biais : 1

(c) Total : 65

Total des paramètres :

1. Poids + biais = 320 + 18496 + 73856 + 8256 + 65 = 100993

4. Entraînement du Modèle
L'entraînement du modèle s'effectue à l'aide de l'optimiseur **Adam**. Cet optimiseur est utilisé pour ajuster les paramètres du modèle en fonction des gradients calculés lors de la rétropropagation. L'optimisation suit un processus en plusieurs étapes :

1. **Propagation Avant (Forward Pass)** : Le jeu de données est passé à travers le modèle. Cela implique que chaque image (histogramme 2D) subit plusieurs convolutions et max-pooling avant d'être aplanie et traitée par les couches fully connected pour obtenir la prédiction du modèle.

2. **Calcul de la Perte et de l'Accuracy** : Une fois que les prédictions ont été faites, la **perte** est calculée en utilisant la fonction **Mean Squared Error (MSE)**, qui mesure la différence entre les valeurs réelles et les prédictions. L'accuracy est calculée avec une marge d'erreur de **0.08**, ce qui permet de déterminer si la prédiction est suffisamment proche de la valeur réelle.

3. **Rétropropagation (Backward Pass)** : Les gradients de la perte par rapport aux poids et aux biais du modèle sont calculés à l'aide de la rétropropagation. Cela permet de déterminer quel impact chaque paramètre a sur l'erreur du modèle, et ainsi ajuster ces paramètres de manière optimale.

4. **Mise à jour des Paramètres** : Après le calcul des gradients, les paramètres du modèle (poids et biais) sont mis à jour par l'optimiseur **Adam**, en fonction de la direction et de l'ampleur des gradients. Ce processus est répété sur plusieurs époques pour améliorer progressivement la performance du modèle.

5. Évaluation et Résultats
À chaque époque, une évaluation est réalisée sur l'ensemble de validation pour mesurer la performance du modèle. L'accuracy et la perte sont calculées sur cet ensemble de validation pour détecter tout surapprentissage (overfitting) ou sous-apprentissage (underfitting) et ajuster les hyperparamètres si nécessaire.
À la fin de l'entraînement, les performances du modèle sont évaluées sur l'ensemble de test. Pour chaque variable cible (par exemple, Open, High, Low, Close), l'accuracy finale est calculée. Les résultats sont ensuite agrégés pour obtenir une estimation globale de la performance du modèle sur l'ensemble de test.
Comment l'accuracy est calculée :

1. **Calcul de l'erreur absolue** :Pour chaque prédiction, on calcule la différence entre la valeur réelle et la valeur prédite, en prenant la valeur absolue pour éviter les écarts négatifs.

$$\text{Erreur absolue} = |\text{Valeur réelle} - \text{Valeur prédite}|$$

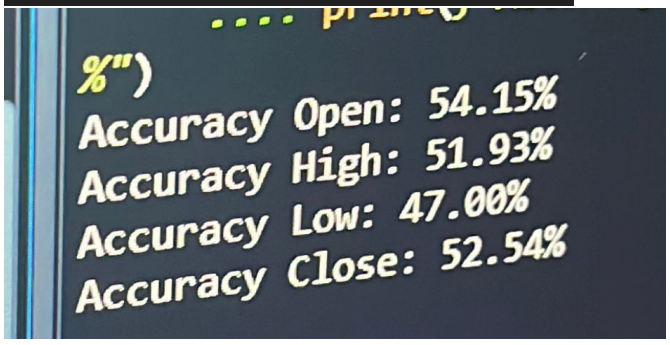
1. **Application de la marge de tolérance** :Une marge fixe (par exemple, 0.8) est définie pour évaluer la précision. Si l'erreur absolue d'une prédiction est inférieure ou égale à cette marge, la prédiction est considérée comme correcte.

$$\text{Condition : } |\text{Valeur réelle} - \text{Valeur prédite}| \leq \text{Marge}$$

1. **Comptabilisation des prédictions correctes** :On détermine le nombre total de prédictions qui respectent cette condition. Cela correspond au nombre de prédictions correctes selon la marge définie.

2. **Calcul de l'accuracy globale** :L'accuracy est ensuite calculée en divisant le nombre de prédictions correctes par le nombre total de prédictions. Cela donne un pourcentage représentant la proportion de prédictions acceptables :

$$\text{Accuracy} = \frac{\text{Nombre de prédictions correctes}}{\text{Nombre total de prédictions}}$$



2.LSTM (pré-entainé) :
On cherche à prédire close
1. Importation des Bibliothèques
Nous avons utilisé des bibliothèques populaires pour la manipulation des données, la normalisation, la création de modèles LSTM et la visualisation des résultats. Cela inclut Numpy, Pandas, Matplotlib, Scikit-learn pour le prétraitement des données, et TensorFlow pour construire et entraîner un modèle de deep learning.
2. Chargement des Données
Le fichier CSV contenant des données boursières est chargé dans un DataFrame Pandas. On explore les colonnes disponibles pour sélectionner celle qui contient les valeurs à prédire (par exemple, les prix de clôture).
3. Prétraitement des Données
Les valeurs de la colonne cible (par exemple, Close) sont normalisées entre 0 et 1 à l'aide de MinMaxScaler. Cela est important pour stabiliser l'entraînement des réseaux neuronaux en réduisant la variance des données.
4. Création de Séquences pour LSTM
Les réseaux LSTM (Long Short-Term Memory) nécessitent des séquences temporelles en entrée. Une fonction est définie pour créer des ensembles de données contenant des séquences de longueur fixe (par exemple, 60 jours consécutifs), utilisées pour prédire la valeur du jour suivant. Ces séquences sont ensuite converties en tableaux NumPy.
5. Division des Données
Les données sont divisées en ensembles d'entraînement et de test pour évaluer la performance du modèle. Typiquement, 80 % des données sont utilisées pour l'entraînement et 20 % pour les tests.
6. Construction du Modèle LSTM
Dans un modèle LSTM (Long Short-Term Memory), les **couches** jouent un rôle crucial pour capturer les relations temporelles dans les données. Dans ce code, plusieurs types de couches sont utilisés dans l'architecture du modèle. Voici une explication de chaque couche dans le modèle :
1. Première couche LSTM (LSTM(50, return_sequences=True))

1. **LSTM (Long Short-Term Memory)** est un type de réseau de neurones récurrent spécialement conçu pour traiter des séquences de données et capturer des dépendances temporelles à long terme.

2. **50** : Ce nombre spécifie le nombre d'unités dans cette couche LSTM. Chaque unité est une cellule mémoire qui apprend et conserve des informations de la séquence passée.

3. **return_sequences=True** : Ce paramètre indique que la couche LSTM renverra les séquences complètes plutôt qu'une seule valeur pour chaque entrée. Cela permet à la couche suivante d'avoir une entrée séquencée. En d'autres termes, cette couche renverra une séquence de valeurs à la couche suivante, ce qui est nécessaire car la couche suivante est également une couche LSTM.

Rôle : Cette couche sert à capturer les dépendances à court terme et à long terme dans les données temporelles (les valeurs passées) et transmet ces informations à la couche suivante.

2. Deuxième couche LSTM (LSTM(50, return_sequences=False))

1. **50** : Comme la première couche, elle a 50 unités. Cela signifie que chaque cellule mémoire dans cette couche peut apprendre et retenir une quantité importante d'informations.

2. **return_sequences=False** : Ce paramètre indique que la couche LSTM renverra **seulement** la dernière sortie de la séquence plutôt que toutes les sorties. Cela signifie que seules les informations pertinentes à la fin de la séquence seront transmises à la couche suivante.

Rôle : Cette couche est utilisée pour affiner et résumer l'information capturée par la couche précédente. Comme elle ne renvoie qu'une seule sortie, elle extrait une caractéristique des séquences passées qu'elle juge importante pour la prédiction future.

3. Couche Dense (Dense(25, activation='relu'))

1. **Dense** : Il s'agit d'une couche complètement connectée, ce qui signifie que chaque neurone de cette couche est connecté à tous les neurones de la couche précédente.

2. **25** : Le nombre de neurones dans cette couche est de 25. Cela permet de capturer des relations plus complexes qui ne peuvent pas être modélisées directement par les couches LSTM.

3. **activation='relu'** : L'activation ReLU (Rectified Linear Unit) est utilisée ici pour introduire une non-linéarité dans le modèle. Elle permet au réseau de mieux capturer des relations complexes dans les données. Elle fonctionne en renvoyant 0 si l'entrée est négative et renvoyant l'entrée elle-même si elle est positive. Cela aide à surmonter les problèmes de vanishing gradients dans les réseaux profonds.

Rôle : Cette couche affine encore la représentation extraite par les couches LSTM et aide à capturer des relations non linéaires plus complexes dans les données. Elle joue un rôle important pour la régulation du modèle et pour une meilleure généralisation.

4. Couche de sortie (Dense(1))

1. **Dense**(**??**) : Cette couche comporte un seul neurone, car la sortie du modèle est une seule valeur, qui représente la prédiction (par exemple, le prix de clôture d'une action).

2. Il n'y a pas d'activation spécifiée ici, ce qui signifie que la sortie est une valeur linéaire.

Rôle : Cette couche fournit la sortie finale du modèle, qui est la prédiction du prix de l'action pour le jour suivant. Le modèle LSTM génère une sortie en fonction des informations temporelles traitées par les couches précédentes.

7. Compilation du Modèle

Le modèle est compilé avec **l'optimiseur Adam**, qui ajuste les poids pendant l'entraînement, et une fonction de perte d'erreur quadratique moyenne (MSE), adaptée aux tâches de régression.

8. Entraînement du Modèle

Le modèle est entraîné avec des données d'entraînement sur 10 epochs , en utilisant un batch size de 32. L'entraînement est suivi par une validation avec l'ensemble de test, permettant d'observer la performance sur des données non vues.

```
Epoch 1/10
5803/5803 ━━━━━━━━━━━━━━━━━━━━ 172s 29ms/step - loss: 2.1114e-04 - val_loss: 1.9161e-05
Epoch 2/10
5803/5803 ━━━━━━━━━━━━━━━━━━━━ 158s 27ms/step - loss: 3.7677e-05 - val_loss: 1.4930e-05
Epoch 3/10
5803/5803 ━━━━━━━━━━━━━━━━━━━━ 164s 28ms/step - loss: 1.3229e-05 - val_loss: 1.5591e-05
Epoch 4/10
5803/5803 ━━━━━━━━━━━━━━━━━━━━ 163s 28ms/step - loss: 3.0955e-05 - val_loss: 1.1010e-05
Epoch 5/10
5803/5803 ━━━━━━━━━━━━━━━━━━━━ 160s 28ms/step - loss: 1.7152e-05 - val_loss: 1.0518e-05
Epoch 6/10
5803/5803 ━━━━━━━━━━━━━━━━━━━━ 159s 27ms/step - loss: 1.7645e-05 - val_loss: 1.4469e-05
Epoch 7/10
5803/5803 ━━━━━━━━━━━━━━━━━━━━ 165s 28ms/step - loss: 1.8862e-05 - val_loss: 1.0665e-05
Epoch 8/10
1100/5803 ━━━━━━━━━━━━━━━━━━━━ 2:04 26ms/step - loss: 6.5816e-06
```

9. Évaluation du Modèle

Après l'entraînement, le modèle est évalué sur les données de test. La perte finale (MSE) est affichée pour mesurer la qualité des prédictions.

```
Perte du modèle : 1.0688507245504297e-05
```

10. Prédictions

Le modèle génère des prédictions pour l'ensemble de test. Les valeurs prédites sont ensuite dénormalisées pour revenir à leur échelle d'origine, facilitant la comparaison avec les données réelles.

11. Calcul de l'Accuracy

Dans le contexte du modèle LSTM pour la prédiction de séries temporelles, l'**accuracy** est une mesure de la qualité de la prédiction du modèle. Cependant, l'accuracy standard (comme celle utilisée pour les tâches de classification) n'est pas directement applicable ici car il s'agit d'un problème de régression (prédire des valeurs continues, comme un prix de clôture). Par conséquent, un autre indicateur est souvent utilisé : le **MAPE (Mean Absolute Percentage Error)**.

MAPE (Mean Absolute Percentage Error) :

Le **MAPE** est une métrique qui mesure l'erreur moyenne entre les valeurs prédites et réelles en pourcentage. Il est utilisé dans les problèmes de régression

pour évaluer la précision des prédictions. Voici la formule pour le calcul du MAPE :

$$\text{MAPE} = \frac{1}{N} \sum_{i=1}^{N} \left| \frac{y_{\text{true},i} - y_{\text{pred},i}}{y_{\text{true},i}} \right| \times 100$$

Où :

1. ytrue,i : la valeur réelle (observée) pour l'échantillon ii,
2. ypred,i: la valeur prédite pour l'échantillon ii,
3. N : le nombre total d'échantillons (par exemple, le nombre d'exemples dans l'ensemble de test).

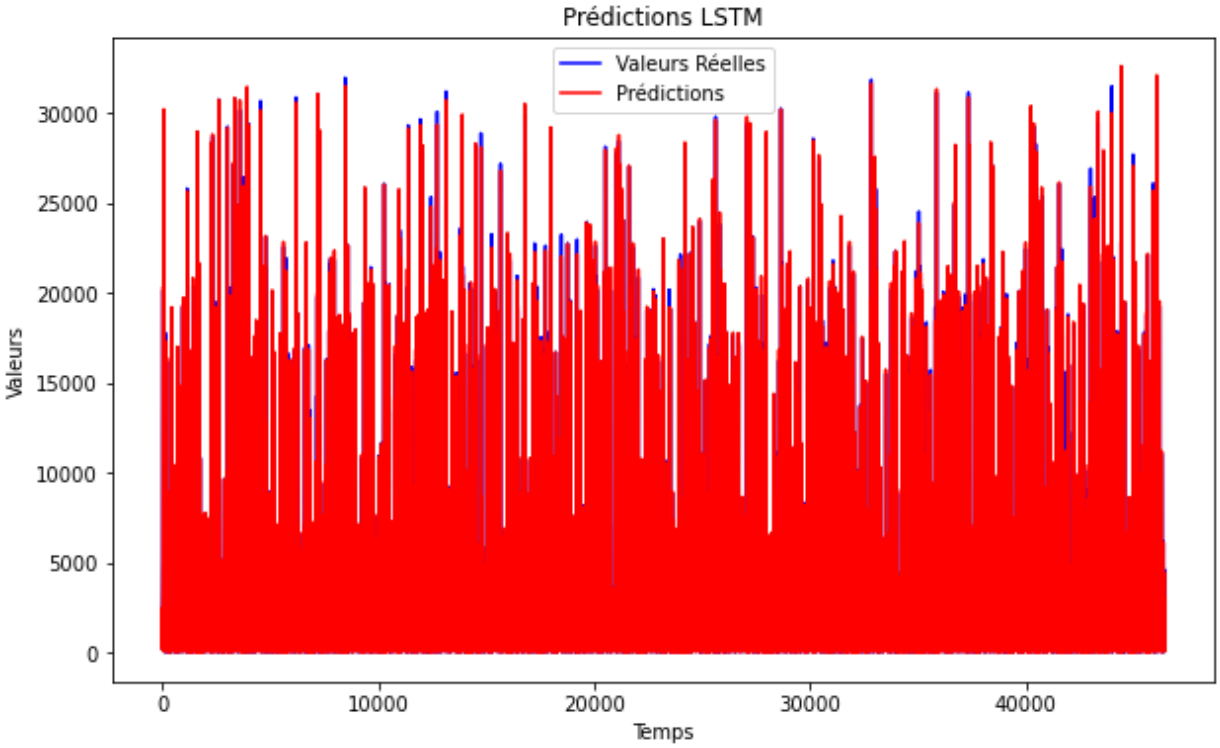
Interprétation du MAPE :

1. Un **MAPE faible** indique que le modèle fait des prédictions très proches des valeurs réelles, donc il a une meilleure précision.
2. Un **MAPE élevé** indique que le modèle fait des erreurs importantes dans ses prédictions, ce qui suggère que les performances du modèle doivent être améliorées.

Accuracies en termes de MAPE :
L'**accuracy** dans ce code est calculée comme :
Accuracy= 100 - MAPE
Cela donne le pourcentage de la précision, c'est-à-dire combien la prédiction est proche des valeurs réelles. Un MAPE de 10 % signifie que l'erreur moyenne est de 10 %, donc l'accuracy sera de 90 %.

```
MAPE : 5.15%
Accuracy : 94.85%
```

12. Visualisation
Les valeurs réelles et prédites sont tracées sur un graphique pour visualiser la capacité du modèle à suivre les tendances des prix. Cela aide à évaluer visuellement la performance du modèle.



Analyse de la Figure :
La figure illustre les résultats des prédictions de CLOSE générées par un modèle LSTM appliqué à des données temporelles, comparées aux valeurs réelles. L'objectif est d'évaluer la capacité du modèle à capturer la dynamique des données.
Interprétation des courbes

1. **Valeurs réelles (en bleu)** : Les points bleus représentent les données observées, considérées comme la vérité terrain.
2. **Prédictions du modèle (en rouge)** : La courbe rouge représente les prédictions continues effectuées par le modèle LSTM.

Observations

1. **Tendance globale** :Le modèle parvient à suivre la tendance globale des données, en capturant les variations majeures des valeurs. Cela démontre que l'architecture LSTM a bien appris les relations temporelles à long terme présentes dans les données.
2. **Variabilité des prédictions** :Les prédictions présentent une forte amplitude et des oscillations importantes, souvent plus marquées que celles des valeurs réelles. Cette caractéristique pourrait indiquer que le modèle amplifie certaines fluctuations présentes dans les données, ce qui pourrait être le signe d'un surajustement.

3. **Densité des données** :En raison de la taille importante du jeu de données, le graphique apparaît dense. Les valeurs réelles (points bleus) sont difficiles à distinguer des prédictions dans certaines zones. Un zoom sur une période spécifique permettrait de mieux évaluer la qualité des prédictions.

Comparaison entre les deux modèles :
Voici une comparaison entre les deux modèles pour la prédiction des valeurs de clôture en bourse : **CNN avec histogrammes 2D** et **LSTM (pré-entraîné)**.
CNN avec histogrammes 2D
Le modèle CNN utilise des images générées à partir d'histogrammes 2D basés sur les relations entre deux jours successifs. Ce modèle analyse les corrélations spatiales dans les histogrammes pour faire des prédictions. Le prétraitement des données consiste à créer ces histogrammes, puis à les transformer en matrices de densité de taille 20x20x1.
L'architecture du modèle CNN inclut plusieurs couches : convolution, max-pooling et couches entièrement connectées. Cette approche permet de capturer les relations spatiales dans les données, mais elle est plus complexe car elle nécessite la génération préalable d'histogrammes.
La performance du modèle dépend fortement de la qualité des histogrammes créés et de la capacité du CNN à extraire des caractéristiques utiles des images. Cependant, cette approche est plus coûteuse en termes de calculs, et l'interprétation des relations spatiales peut parfois être difficile.
Les prédictions générées par ce modèle sont des valeurs continues de clôture pour les jours suivants, mais la complexité computationnelle élevée et la difficulté de gérer de très grandes quantités de données peuvent constituer des limites.
LSTM (pré-entraîné)
Le modèle LSTM, quant à lui, traite directement les séquences temporelles numériques, comme les prix des 60 derniers jours. Les données sont prétraitées en les normalisant entre 0 et 1 à l'aide de MinMaxScaler, ce qui permet d'améliorer la stabilité du modèle lors de l'entraînement.
L'architecture du modèle LSTM comprend plusieurs couches LSTM (long-term et short-term memory) suivies de couches denses. Cela permet au modèle d'apprendre les dépendances temporelles longues et courtes dans les séries de données. Cette approche est généralement moins coûteuse en termes de calculs que le modèle CNN, car elle n'implique pas la transformation des données en images.
Les performances du modèle dépendent principalement de la préparation des données et de la régularisation du modèle. LSTM est très efficace pour capturer les relations temporelles dans des séries de données continues comme les prix boursiers, et il est bien adapté aux grands ensembles de données numériques. Toutefois, si le modèle n'est pas bien régularisé, il peut souffrir de surajustement.
Conclusion
Le modèle CNN avec histogrammes 2D peut être adapté si vous souhaitez exploiter des relations spécifiques entre les jours sous forme visuelle, mais

il présente une complexité computationnelle plus élevée et nécessite une interprétation des relations spatiales. En revanche, le modèle LSTM est généralement plus adapté aux séries temporelles classiques et permet de capturer directement les dépendances temporelles. Pour des données boursières, LSTM est souvent plus performant, sauf si des relations visuelles non évidentes dans les données nécessitent l'approche CNN.