**技術指標的應用**

**OHLC** 是股票或其他金融資產的價格數據表示方式，主要由四個指標組成，分別是：

1. **Open**（開盤價）：指交易時段開始時的第一筆成交價格。
2. **High**（最高價）：指交易時段內的最高成交價格。
3. **Low**（最低價）：指交易時段內的最低成交價格。
4. **Close**（收盤價）：指交易時段結束時的最後一筆成交價格。

一般而言，這四個指標會與成交量**Volume一起使用**

**Volume**（成交量）是指在指定時間內某一資產被交易的總數量。

### ****為什麼單純使用 OHLC 而不用 Volume？****

1. **聚焦於價格變化的模式**：
   * **核心資訊來源**：OHLC 本身已包含完整的價格趨勢信息，例如趨勢的方向、波動範圍，由於 **Volume並不直接反應在價格的K線上，可能會導致機器學習曾將額外的數據噪音，模型可能會過度擬合。**
2. **模型的特性匹配**：
   * 因為使用的是圖像生成方法（OHLC黑白圖像），圖像主要捕捉價格形態和波動模式，而成交量的訊息較難在黑白圖像中直接呈現。

### 稀疏矩陣抽象資料類型（Sparse Matrix Abstract Data Type）

**ADTSparseMatrix** 是：

* **物件**：由一組三元組 組成，其中：
  + row 和 column 是整數，並且每個組合都是唯一的。
  + value 來自集合 item。
* **函數**：

對於所有

1. **SparseMatrix Create(maxRow, maxCol)**：返回一個稀疏矩陣，該矩陣最多可以容納

個元素，且最大列數為 maxRow，最大行數為 maxCol。

1. **SparseMatrix Transpose(a)**：
   * 返回通過將稀疏矩陣 a 中每個三元組的列（row）和行（column）值互換後生成的矩陣。
2. **SparseMatrix Add(a, b)**：
   * 若 a 和 b 的維度相同：
     + 返回由 a 和 b 對應位置的元素相加所生成的矩陣，即具有相同行列值的元素進行相加。
   * 否則返回錯誤。
3. **SparseMatrix Multiply(a, b)**：
   * 若 a 的列數等於 b 的行數：
     + 返回矩陣 d，該矩陣通過以下公式計算得出：

其中，d[i][j] 是矩陣中第i列、第j 行的元素。

* + 否則返回錯誤。

### ****稀疏矩陣的使用****

在程式中，OHLC 圖像的生成本質上是一種稀疏矩陣操作：

1. **黑白圖像的稀疏性**：
   * 生成的圖像初始化為 np.zeros(img\_size, dtype=np.uint8)，是一個全黑（像素值為 0）的矩陣。
   * 只有在特定的位置（例如 High、Low、Open、Close 價格所對應的像素位置）被畫上白色線條（像素值為 255）。
   * 這種稀疏性大部分矩陣元素的值為 0，符合稀疏矩陣的定義。
2. **圖像數據的高效處理**：
   * 將價格數據映射到圖像上時，程式僅在指定位置進行繪圖操作（使用 cv2.line 畫出 High 和 Low 的直線），大大降低了非必要像素點的計算。
   * CNN 模型能有效地從這種稀疏矩陣中提取關鍵模式，避免對冗餘訊息的處理。

### ****移動窗口的使用****

1. **窗口的概念**：
   * 程式使用一個固定大小的移動窗口（WINDOW\_SIZE = 30），表示一次只考慮 30 天的 OHLC 數據來生成圖像。
   * 隨著窗口向前移動（從 i 到 i+WINDOW\_SIZE），每次產生一張新的圖像，確保時間序列的連續性。
2. **應用場景**：
   * 移動窗口能捕捉市場在短期內的價格模式，特別適合用於識別趨勢和反轉信號。
   * **窗口長度的選擇**：
     + WINDOW\_SIZE 的長度會影響模型的輸入特徵：
       - **較短窗口（如 10 天）**：可能捕捉短期波動，但容易忽略長期趨勢。
       - **較長窗口（如 60 天）**：更適合捕捉中長期趨勢，但可能會稀釋短期波動的重要性。
     + 本程式選擇 30 天作為平衡短期與中期趨勢的窗口大小。
3. **窗口數據的標準化**：
   * 在窗口內，數據進行了高低價的縮放和標準化，使其適合圖像的縱軸範圍（img\_size[0]-1），這種處理方法消除了不同資產之間價格級別的影響。

程式中，核心技術指標應用集中於將股票市場數據（OHLC格式）轉換為黑白圖像，結合卷積神經網路（CNN）模型進行未來收益率的預測，具體如下：

**1. 指標的選取與數據處理**

* **數據標準化**：利用 MinMaxScaler 將 Open、High、Low、Close 和 Volume 等價格與成交量數據進行標準化處理，確保不同範圍的數據在統一尺度上進行分析，避免數值差異過大影響模型性能。
* **OHLC 圖像生成**：
  + 將固定窗口（30天）的市場數據轉換為64x64像素的黑白圖像。
  + 使用 cv2.line 繪製當前窗口內的 High、Low、Open 和 Close 價格，將這些價格動態表示為圖像中的幾何形狀。
  + 僅保留價格變化的空間模式特徵。

1. **CNN 模型的設計與應用**

這段程式碼設計了一個用於處理 **2D 圖像數據**並進行迴歸預測的 **卷積神經網絡（CNN）**，架構包含卷積層、激活函數、池化層以及全連接層。以下是詳細的設計解釋：

該模型的目的是從 OHLC 圖像中提取價格變化的空間模式特徵，並基於這些特徵進行未來收益率的預測（迴歸任務）。

### ****模型的基本架構****

#### ****1. 抓取圖像細節的部分（卷積層）****

* **作用**：如同人類看圖一樣，這部分是幫助模型從圖像中抓取關鍵的細節，例如圖中的趨勢形狀或價格的波動模式。
* **怎麼做**：
  + 用小的「掃描器」（稱為卷積核，大小為3X3）在圖像上掃描，找出圖中的局部特徵。
  + 每掃描一次，就得到一組新圖，這些圖包含不同的模式（例如價格上升、下降等）。
  + 每次掃描之後，把圖像縮小一點，方便重點關注更重要的部分。

#### ****2. 把重要信息壓縮進腦袋的部分（全連接層）****

* **作用**：把前面提取的圖像訊息，整理成數字，並進一步學習怎麼把這些數字和收益率連結起來。
* **怎麼做**：
  + 先把圖像數據壓平成一列，變成簡單的數字排列。
  + 用一層「神經網路」來學習數字之間的關係。
  + 最終輸出一個數字，代表預測的收益率。

#### ****3. 避免過度自信的部分（Dropout 層）****

* **作用**：讓模型在學習時，不要太依賴某些特定的細節，避免「記住答案」而不是學會預測規律。
* **怎麼做**：在訓練時，隨機遮住一部分神經網路的連線，讓模型學得更全面。

### ****Dropout 層的介紹****

**Dropout** 是一種在神經網路中使用的正則化技術，目的是減少模型的過度擬合，從而提升模型的泛化能力。Dropout 是在訓練過程中隨機屏蔽（丟棄）部分神經元，讓模型在學習時不過度依賴某些特定的特徵。

### ****Dropout 的工作原理****

1. **訓練時的操作**：
   * 在每一次前向傳遞（forward pass）中，隨機選擇一定比例的神經元，使其暫時停止工作（即輸出值設為 0）。
   * 被屏蔽的神經元不參與這一次的計算，也不影響梯度的更新。
2. **測試時的操作**：
   * 在測試過程中，Dropout 不會屏蔽任何神經元，而是根據訓練時的丟棄比例縮放神經元的輸出值，保證整體輸出的一致性。

### ****Dropout 的參數****

1. **丟棄比例（）**：
   * Dropout 層的主要參數是丟棄比例，通常設置為 0.2 ~ 0.5。
   * 較大的值（如 0.5）適合更複雜的模型，可以強化正則化效果。
   * 較小的值（如 0.2）適合較小的模型，避免過度削弱模型能力。

### ****Dropout 的優勢****

1. **減少過度擬合**：
   * Dropout 強迫模型在每次訓練時僅依賴部分神經元，減少對特定特徵的過度依賴，讓模型學習到更普遍的規律。
2. **提升模型的泛化能力**：
   * 由於 Dropout 的隨機性，模型能更好地適應未知數據，從而提高在測試集上的性能。
3. **簡單且高效**：
   * Dropout 的實現和計算量都很簡單，但能顯著提升模型性能。

### ****Dropout 的使用場景****

1. **在全連接層中使用**：
   * Dropout 通常應用於全連接層，因為這些層容易過度擬合。
   * 在 CNN 中，Dropout 通常放在特徵提取階段之後，用於高層次特徵的正則化。
2. **不適合卷積層**：
   * 卷積層的參數數量相對少，Dropout 的正則化效果有限，因此不常用於卷積層。

### ****這個模型是怎麼工作的？****

1. **輸入數據**：一張 64X64 的 OHLC 圖像（表示股票價格的變化）。
2. **提取特徵**：模型的三層卷積會不斷提取圖像的細節，從基礎的價格變化線條到更複雜的趨勢形狀。
3. **整理數據**：把所有的圖像特徵壓縮成簡單的數字，然後用數學公式計算出收益率。
4. **輸出結果**：最後輸出一個數字，代表模型預測的未來收益率（例如預測未來可能賺 5% 或虧 3%）。

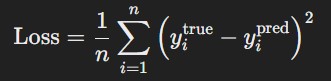
### ****Epoch :****

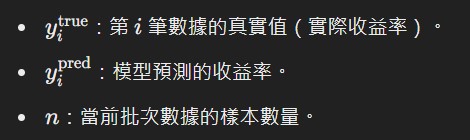
**定義**：Epoch 是模型在訓練數據集上完成一次完整遍歷的過程。如果數據集有 1000 筆資料，而每次訓練處理 32 筆（批次大小，batch size = 32），那麼需要 ⌈1000/32⌉=32 次更新完成一個 **Epoch**。

### ****Loss****：

Loss（損失）是一個數值，用於衡量模型預測結果與真實目標值之間的差距。

* + 在這裡，使用的是 **均方誤差（MSE, Mean Squared Error）損失函數**，公式如下：





### ****Python使用套件 :****

### ****PyTorch****

PyTorch 是一個強大的深度學習框架，適合構建和訓練各種神經網絡模型。

* **主要用途**：
  + 用於建構神經網路模型、張量操作和高效的 GPU 計算。
  + 提供豐富的工具包來處理深度學習相關的任務。
* **常用功能**：
  + **張量（Tensor）操作**：高效處理多維數據
  + **GPU 加速**：自動將計算分配到 CUDA 支援的 GPU 上。

### ****2.**** torch.nn****（神經網絡模組）****

這是 PyTorch 的一部分，專門用於建構和定義神經網絡模型。

* **主要用途**：
  + 提供構建神經網絡所需的模塊（如卷積層、全連接層、激活函數等）。
* **常用模組**：
  + nn.Conv2d：2D 卷積層，用於處理圖像數據。
  + nn.Linear：全連接層，用於處理壓平後的數據。
  + nn.ReLU：激活函數，增加非線性表現能力。
  + nn.Sequential：組合多個層，形成一個完整的網絡。

### ****3.**** torch.optim****（優化器模組）****

PyTorch 提供的優化器，用於調整模型參數以降低損失函數。

* **主要用途**：
  + 實現反向傳播後的參數更新。
* **常用優化器**：
  + optim.SGD：隨機梯度下降法。
  + optim.Adam：自適應梯度優化器，收斂速度快，適合深度學習。

### ****4.**** torch.utils.data****（數據處理模組）****

PyTorch 的數據處理工具，用於處理和加載訓練數據。

* **主要用途**：
  + 定義數據集（Dataset）和數據加載器（DataLoader）。
* **常用類別**：
  + Dataset：自定義數據集，包含數據的讀取和預處理邏輯。
  + DataLoader：批量加載數據，支持多線程和隨機打亂數據。

### ****5.**** yfinance****（股票數據下載）****

yfinance 是一個方便的工具，用於從 Yahoo Finance 獲取股票市場數據。

* **主要用途**：
  + 下載歷史股票數據，如開盤價（Open）、收盤價（Close）、最高價（High）、最低價（Low）、成交量（Volume）。
* **使用方法**：
  + yf.download()：下載指定股票的歷史數據，支持設定日期範圍。

### ****6.**** sklearn.preprocessing****（數據預處理工具）****

Scikit-learn 提供的數據預處理模組，用於標準化和縮放數據。

* **主要用途**：
  + 使數據符合模型的輸入要求，提升模型的訓練效果。
* **常用工具**：
  + MinMaxScaler：將數據縮放到 [0, 1] 範圍內，適合處理價格數據。

### ****7**** cv2****（OpenCV）****

OpenCV 是一個開源的計算機視覺和圖像處理庫。

* **主要用途**：
  + 用於生成和處理 OHLC 黑白圖像。
* **常用功能**：
  + cv2.line()：在圖像上畫線。
  + cv2.imshow()：顯示圖像。

輸入輸出範例:

請輸入股票代號: TSLA

[\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*100%%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*] 1 of 1 completed

請輸入訓練的次數: 50

Epoch 1, Loss: 219.7012

Epoch 2, Loss: 0.1248

Epoch 3, Loss: 0.1206

Epoch 4, Loss: 0.1149

Epoch 5, Loss: 0.1067

Epoch 6, Loss: 0.0939

Epoch 7, Loss: 0.0880

Epoch 8, Loss: 0.0809

Epoch 9, Loss: 0.0854

Epoch 10, Loss: 0.0671

Epoch 11, Loss: 0.0731

Epoch 12, Loss: 0.0661

Epoch 13, Loss: 0.0646

Epoch 14, Loss: 0.0686

Epoch 15, Loss: 0.0629

Epoch 16, Loss: 0.0621

Epoch 17, Loss: 0.0607

Epoch 18, Loss: 0.0592

Epoch 19, Loss: 0.0564

Epoch 20, Loss: 0.0570

Epoch 21, Loss: 0.0571

Epoch 22, Loss: 0.0598

Epoch 23, Loss: 0.0615

Epoch 24, Loss: 0.0649

Epoch 25, Loss: 0.0587

Epoch 26, Loss: 0.0573

Epoch 27, Loss: 0.0512

Epoch 28, Loss: 0.0548

Epoch 29, Loss: 0.0561

Epoch 30, Loss: 0.0552

Epoch 31, Loss: 0.0648

Epoch 32, Loss: 0.0518

Epoch 33, Loss: 0.0532

Epoch 34, Loss: 0.0487

Epoch 35, Loss: 0.0496

Epoch 36, Loss: 0.0579

Epoch 37, Loss: 0.0573

Epoch 38, Loss: 0.0554

Epoch 39, Loss: 0.0545

Epoch 40, Loss: 0.0533

Epoch 41, Loss: 0.0582

Epoch 42, Loss: 0.0502

Epoch 43, Loss: 0.0506

Epoch 44, Loss: 0.0478

Epoch 45, Loss: 0.0487

Epoch 46, Loss: 0.0480

Epoch 47, Loss: 0.0527

Epoch 48, Loss: 0.0445

Epoch 49, Loss: 0.0444

Epoch 50, Loss: 0.0462

模型參數已保存至 'monthly\_strategy\_model\_weights.pth'

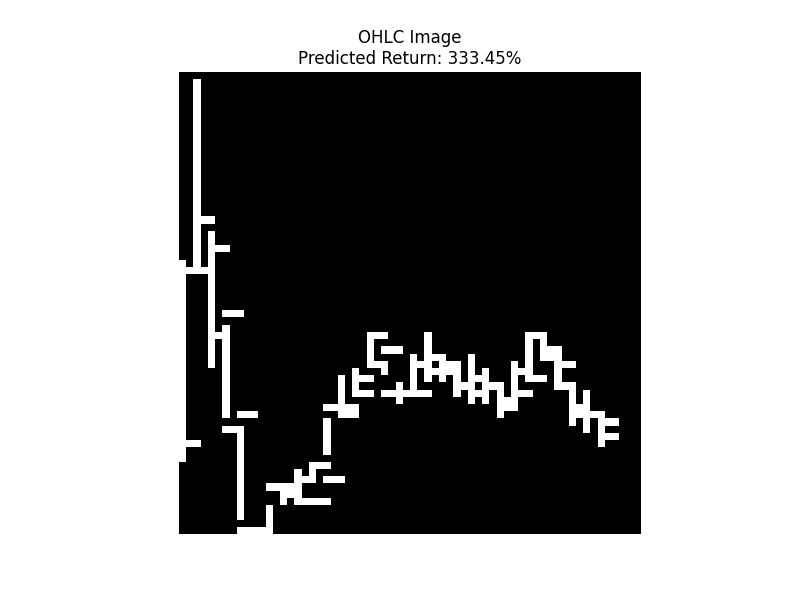
c:\Users\Leon\Desktop\程式語言資料\python\金融科技學程\_程式設計與應用\期末報告\OHLC8.py:131: FutureWarning: You are using `torch.load` with `weights\_only=False` (the current default value), which uses the default pickle module implicitly. It is possible to construct malicious pickle data which will execute arbitrary code during unpickling (See https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-models for more details). In a future release, the default value for `weights\_only` will be flipped to `True`. This limits the functions that could be executed during unpickling. Arbitrary objects will no longer be allowed to be loaded via this mode unless they are explicitly allowlisted by the user via `torch.serialization.add\_safe\_globals`. We recommend you start setting `weights\_only=True` for any use case where you don't have full control of the loaded file. Please open an issue on GitHub for any issues related to this experimental feature.

loaded\_model.load\_state\_dict(torch.load("monthly\_strategy\_model\_weights.pth", map\_location=device))

模型參數已成功加載

測試輸入的預測值: 0.1300

預期年化報酬率: 333.45%



參考資料 :

Bender, J., Briand, R., Subramanian, R. A., & Nielsen, F. (2021). Decarbonization Factors: A Framework for Assessing Corporate Net-Zero Progress. SSRN. <https://doi.org/10.2139/ssrn.3756587>