QUANTITATIVE ISSUE

2026. 1. 8

MACRO팀

김동영, CFA Quant Analyst
dy76.kim@samsung.com

리서치센터 리포트
바로가기

퀀트 모델링 A to Z

(7) CNN AI 투자 모델 (기술적 분석 기반)

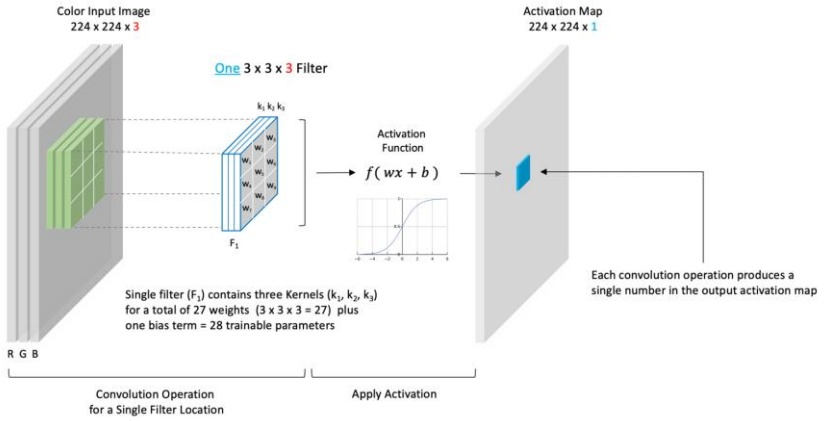
- 금융 시계열 처리에 강점을 가진 CNN 모델에 대한 설명과, 퀀트 모델 접목 방법을 정리

WHAT'S THE STORY?

CNN(Convolutional Neural Network, 합성곱 신경망)은, 공간적 구조 혹은 시계열적 구조가 중요한 데이터에서 국소 패턴을 자동으로 학습하는 딥러닝 모델이다. 이미지 인식 및 처리에서 가장 기본으로 사용하는 모델이며, 시계열 패턴 감지 장점으로 인해서 금융 분야에서도 유용한 AI 모델이다.

본 자료는, CNN 모델에 대한 이해와 사용 방법, 그리고 퀀트 투자 모델에 접목하는 방법에 대해서 예시를 통해서 상세히 설명하고자 한다.

CNN 모델의 합성곱 처리 개념도



자료: https://www.oksim.ua/understanding-convolutional-neural-networks-cnns/

삼성증권

I. CNN 모델 구조

본 리포트는, 딥러닝의 주요 모델 중 하나인 CNN 모델을 소개하고, 이를 퀀트 투자 모델에 접목하는 방법을 예시를 통해 구체적으로 설명하고자 한다.

이번의 "CNN 모델"과 시리즈 직전 리포트의 "MLP 모델"은, 모두 딥러닝 모델이란 공통점을 가지고 있다. 딥러닝 모델의 일반적인 이해 및 사용법에 대해서는, 26/1/2일 자 "퀀트 모델링 A to Z - (6) MLP(Multi-Layer Perceptron) AI 투자 모델" 리포트에 먼저 설명되어 있다.

여기서 사용 및 설명하는 전체 코드의 프로그래밍 환경은 다음과 같다.

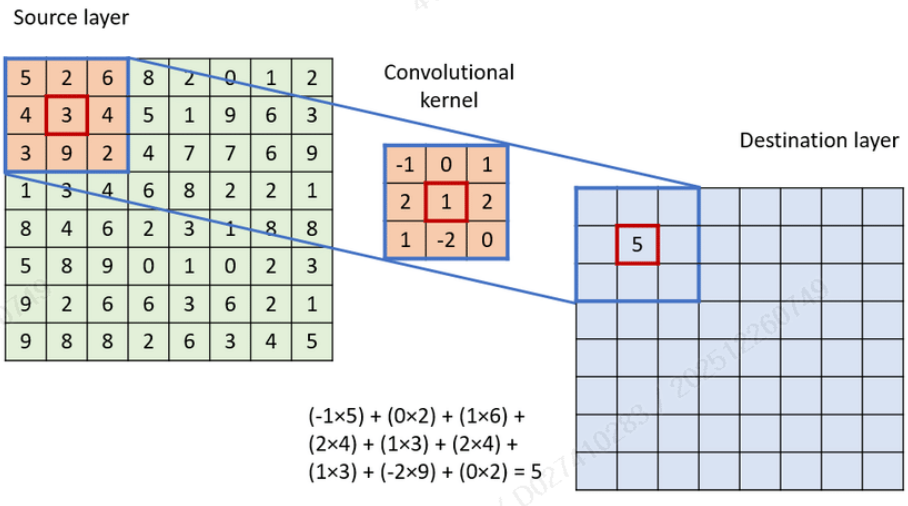
```
Python 3.12.7
tensorflow 2.20.0
tensorflow.keras 3.12.0
```

1) CNN의 이해

CNN(Convolutional Neural Network, 합성곱 신경망)은 이미지나 시계열 데이터에서 패턴을 자동으로 찾아내는 데 특화된 신경망이다. 요즘의 휴대폰 사진 앱에서 많이 사용하는 AI 기능(인물 탐지)에, 항상 기초로 들어가는 알고리즘이라 할 수 있다.

우선 CNN에서 사용되는 "합성곱" 연산이란, 작은 필터(커널)를 데이터 위에서 이동시키며 곱하고 더해, 특징을 뽑아내는 연산이다. 이미지 데이터 전체에 대해서 필터를 이동시켜 가며, 각 위치마다 출력값을 만들어 내어, 최종적인 출력 데이터를 생성한다. 실제 계산은 1) 작은 행렬(필터)를 입력 위에 얹고, 2) 겹치는 자리끼리 곱하고, 3) 그 곱한 값을 전부 더한 다음, 4) 그 합을 그 위치의 "점수"로 기록하고, 5) 이 일련의 과정을 필터를 계속 옮기면서 전체 영역에 대해서 반복하는 방식이다(아래 개념도를 참고).

합성곱(Convolution) 계산법



자료: ResearchGate

이 연산은, 국소 영역을 보면서 패턴을 감지하는 계산이라고 할 수 있다. 이미지의 경우에는, 픽셀 수(데이터 수)가 상당히 많고 한 픽셀과 인접한 픽셀 간의 위치 관계(인접한 점, 멀리 떨어진 점)가 존재한다. 따라서, 이미지 데이터를 단순히 단순 신경망 모델에 넣으면, 파라미터 수가 너무 많아지고, 또 위치 정보가 사라지는 단점이 있다. CNN 구조에서는, 작은 행렬인 "필터"가 이미지 위를 슬라이딩하면서 계산하여, "특성 맵(feature map)"이라고 하는 출력을 만들어낸다. 이 특성 맵은, 이미지에서의 경계, 외곽선, 모서리 같은 단순한 패턴을 감지할 수 있다. 이 필터를 여러 개 적용해 특성 맵을 여러 개 만들고, 이를 다시 정보를 축약시키는 과정을 통해 최종 타깃까지의 신경망을 연결하는 구조를 만드는 것이 CNN 모델의 방식이다. 또한 합성곱은 필터(행렬) 하나를 데이터의 모든 영역에 계산 적용하므로, 사용 파라미터 수를 크게 줄이는 장점도 있다.

합성곱 연산을 비유해서 설명하자면, 돋보기(필터)로 그림을 조금씩 보면서 "여기에 직선이 있다", "여기에 경계선이 있다" 등의 특징을 기록하는 방식이다.

합성곱의 "필터"는, 사실상 어도비 포토샵에서 사용하는 "필터"와 같은 어원이고 동일한 역할을 한다. 포토샵의 필터는, 이미지의 각 픽셀에 대해 동일한 행렬 연산을 적용해 흐릿하게 만들거나, 뚜렷하게 만들거나, 외곽선을 강조하거나 하는 이미지(정보) 변환의 역할을 한다(Blur, Sharpen, Sepia, Edge Detection 기능 등).

예) 세로 윤곽선 탐지 필터

1	0	-1
1	0	-1
1	0	-1

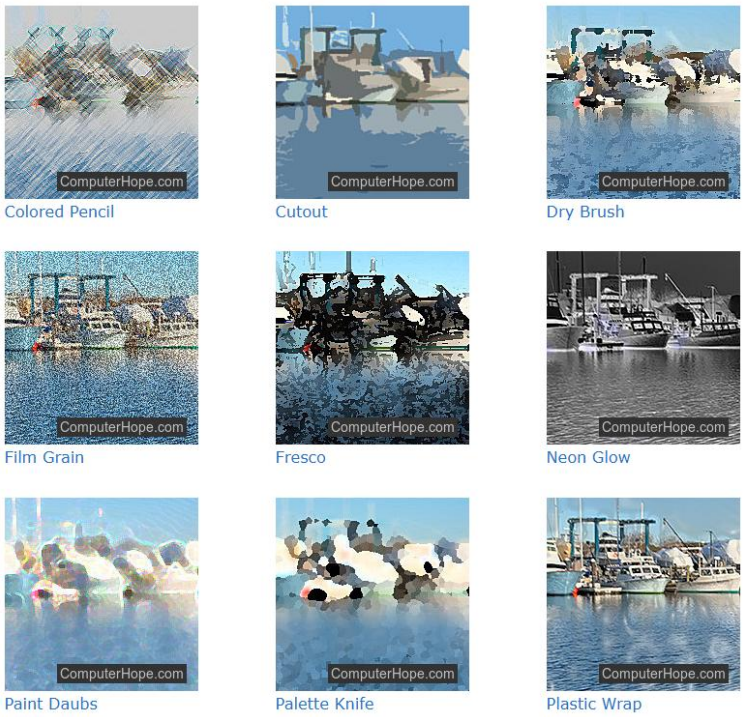
예) 가로 윤곽선 탐지 필터

1	1	1
0	0	0
-1	-1	-1

예를 들어, 위 왼쪽에 나와 있는 "세로 윤곽선 탐지 필터"를 이미지에 적용하면, 적용 후 이미지는 세로 윤곽선이 크게 부각되는 형태로 바뀐다. 왜냐하면 원래 그림의 "특정 세로줄 영역"이 자신의 왼쪽 선 부분과 오른쪽 선 부분의 색상(수치)이 많이 다르다면, 해당 필터 적용 후에는 그 세로줄 영역의 수치(색상)가 크게 튀도록 나오면서, 출력 데이터상에서는 해당 영역이 강하게 눈에 띄기 때문이다.

마찬가지로, 위 오른쪽에 나온 "가로 윤곽선 탐지 필터"를 이미지에 적용하면, 가로 윤곽선이 부각되는 형태가 나타나게 된다. 필터의 역할은, 이런 방식으로 입력 데이터에서의 특정 패턴을 감지하는(feature detection) 역할을 담당한다. 여기에 층을 계속 더 쌓으면, 단순한 패턴이 아닌 고차원의 패턴을 감지하는 역할이 생길 수 있다.

이미지 편집 앱의 필터 기능



자료: https://www.computerhope.com/jargon/p/photoshop-filters.htm

CNN 모델에의 작동은, 이런 "필터" 행렬의 수치를 미리 지정하는 것이 아니고, '무작위 값'으로 채운 필터로 시작한 후, 학습을 반복하면서 각 필터(행렬) 수치가 역전파에 의해서 수치가 조정되는 형태다.

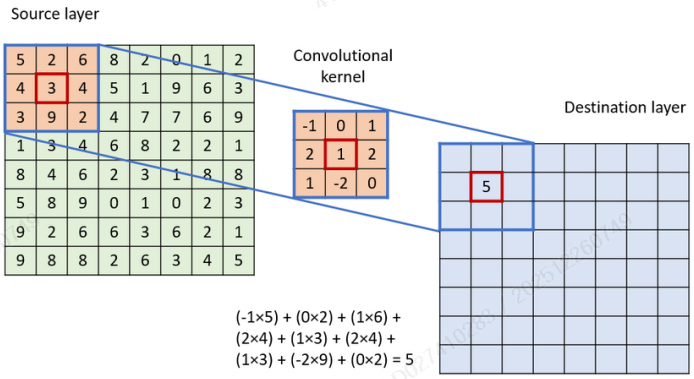
즉, 예를 들어, 입력에 "이미지"를 넣고, 출력에 "세로 윤곽선 개수를 카운트한 데이터"를 가지고 계속 학습을 하게 되면, 초기 합성곱 계층의 한 필터는 세로 윤곽선 탐지에 특화된 행렬로 변하게 되는 셈이다. 입력에 이미지를 넣고, 출력에 "개 혹은 고양이"를 넣은 데이터를 가지고 계속 학습을 하게 되면, CNN 내의 여러 필터들이 개, 고양이 구별에 특화되는 수치 행렬의 필터들로 변화하게 된다. 복잡한 작업을 수행하기 위해서는, 당연히 필터가 1개만 있어서는 안 된다. CNN에서는 항상 여러 개의 필터를 사용하여, 이 여러 필터들이 여러 패턴을 각기 따로 탐지한다.

2) CNN의 데이터 구조

모델의 구조를 잘 이해했는지는, 데이터의 차원과 파라미터 수 계산을 통해서 정확히 알 수 있다. 이에 대해 정리해 보자.

흑백 이미지 + 2차원 합성곱의 데이터 구조

흑백 이미지에 대한 합성곱 계산 개념도



자료: ResearchGate

위의 예시에서는 그림 이미지의 데이터 표현을 2차원 행렬로 표시했다. 사실상 이는 흑백 그림일 때만 가능하다. 흑백 이미지 한 장은 [Height, Width, 1]의 데이터 구조(shape)를 가진다. 높이와 너비만큼 픽셀 수가 존재하고, 각 픽셀에 대해서 '흑에서 백 사이의 명도'를 나타내는 수치가 하나씩 들어간다.

2차원 데이터 위를 이동하는 합성곱 연산에는 2차원 행렬의 필터가 사용된다. 즉, 이때 한 개 필터는 [K1, K2]의 행렬인 것이다. 이 한 개 필터를 결정하는 데 사용되는 파라미터 수는, 행렬 원소 수인 [K1 * K2]와 절편 파라미터 수 1개를 더한, "K1 * K2 + 1"개가 된다.

(이때, 실제로는 필터 행렬에서 가로와 세로 크기를 주로 동일하게 설정함.)

한편, 합성곱 연산 시에는 padding="same"이라고 해서, 합성곱 계산 후의 출력 데이터 모양이 최초 이미지 데이터와 같도록 0으로 테두리를 채운 후에 계산하는 기법이 보통 사용된다. 이 경우, 입력 데이터의 데이터 형태(가로, 세로 크기)와 연산 후 출력 데이터의 데이터 형태(가로, 세로 크기)는 같게 된다. 즉, 합성곱 연산 후의 출력 결과는 [Height, Width, 1]의 데이터 구조를 유지한다.

이를 정리하면 다음과 같다.

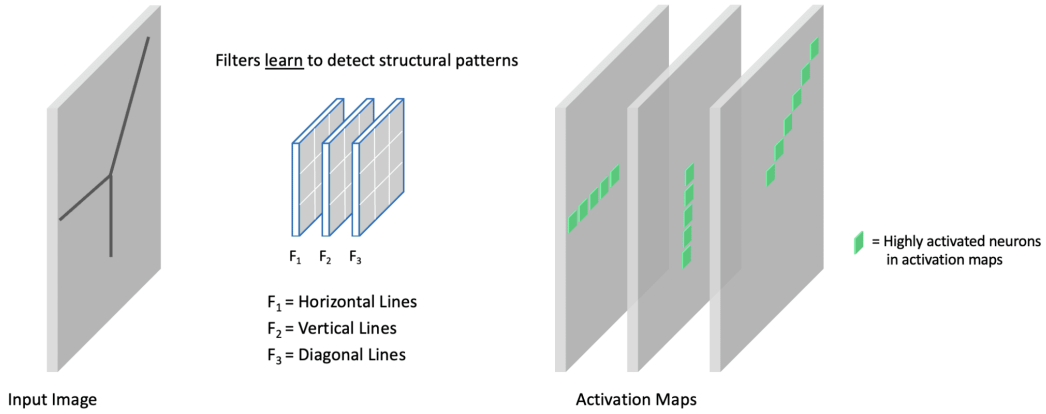
입력 데이터:	**흑백 이미지 한 장. [H, W, 1] 차원**
필터 형태:	**[K1, K2] 모양의 필터 1개**
→	
필터 파라미터 수:	**K1 * K2 + 1**
출력 데이터:	**피쳐 맵 한 장. [H, W, 1] 차원**

padding="same", strides=1 가정 시

보통의 경우, 필터를 여러 개 사용해서, 이미지상의 여러 가지 특성을 각각 찾아내는 방식을 쓴다. 이렇게, 필터를 F개 사용하는 CNN 계층에서는, 데이터 차원 및 파라미터 수는 아래와 같이 변화한다.

입력 데이터:	흑백 이미지 한 장. [H, W, 1] 차원
필터 형태:	[K1, K2] 모양의 필터, F개가 있음
→	
필터 파라미터 수:	(K1 * K2 + 1) * F
출력 데이터:	F개의 피쳐 맵. [H, W, F] 차원

흑백 이미지 한 장에, 여러 개의 필터를 사용한 합성곱 계산 개념도



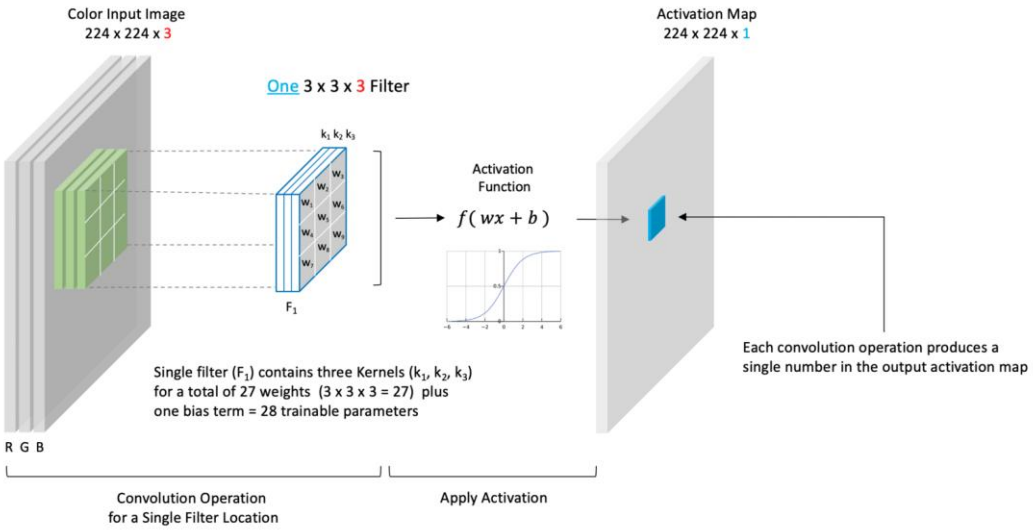
자료: https://medium.com/@mmaharaja

현실 세계의 CNN 모델 사용 시에는, 여러 장의 이미지 데이터를 가지고 학습한다. 따라서, 실제 입력 데이터는 [H, W, 1] 차원의 이미지가 N개(Number of samples)가 모여 있는 형태가 된다. 흑백 이미지를 사용할 때의 현실적인 CNN 계층의 구조는 다음과 같다.

실제 입력 데이터:	흑백 이미지 여러 장. [N, H, W, 1] 차원
필터 형태:	[K1, K2] 모양의 필터, F개가 있음
→	
필터 파라미터 수:	(K1 * K2 + 1) * F
실제 출력 데이터:	이미지별 F개의 피쳐 맵. [N, H, W, F] 차원

컬러 이미지 + 2차원 합성곱의 데이터 구조

컬러 이미지에 대한 합성곱 계산



자료: https://www.oksim.ua/understanding-convolutional-neural-networks-cnns/

실제로 일반적인 그림은 색상 정보가 있기 때문에, "한 장의 이미지"가 2차원 행렬이 아닌 3차원 텐서 데이터로 표현된다. RGB 색상 정보를 가지는 그림 이미지는, 보통 [픽셀 높이, 픽셀 너비, 3(RGB 각각의 수치 정보)] 차원의 배열 데이터로 표현이 된다.

이미지 편집에서 "RGB 채널"이란 용어가 사용되듯이, CNN에서도 채널이란 용어가 사용된다. CNN에서의 채널은 "같은 위치 정보를 공유하는 여러 겹의 특성 층"이라고 할 수 있다. 예로, 컬러 이미지 파일에서의 채널은 빨강의 농도, 녹색의 농도, 파랑의 농도인 RGB 3개 채널이 존재한다. 주식 투자와 관련한 데이터를 생각해 보면, 일별 주가, 일별 거래량, 일별 시가 등이 서로 다른 지표들이 각각 하나의 채널로 작용한다.

컬러 이미지 한 장의 데이터 구조는 [H, W, C] 차원이 된다(일반적인 RGB 채널만 있으면, C=3이 됨). 그리고, 통상적인 CNN 모델에서는 한 개 "필터"가 입력의 모든 채널을 동시에 보고, 같이 계산하는 방식이다. 채널 간의 상호 작용을 같이 반영하기 위해서, 한 개 "필터" 계산 시, 한 국소 영역 부분에서 컬러 이미지의 R값, G값, B값을 동시에 행렬 계산에 넣어서 하나의 값만 출력하는 방식이다. 따라서, 2차원 '컬러' 이미지에 대한 "합성곱 필터"의 데이터 구조는, 채널 수만큼 한 차원이 늘어나서 [K1, K2, C] 형태의 행렬을 띠게 된다.

(이때의 필터는 '가로 K1 칸, 세로 K2 칸, 깊이 C 칸의 3차원 텐서'가 되어서, 합성곱 계산 후의 출력 데이터는 가로, 세로 구조만 남기고, 깊이는 1로 줄여 버림)

입력 데이터:	컬러 이미지 한 장. [H, W, C] 차원
필터 형태:	[K1, K2, C] 모양의 필터 1개
→	
필터 파라미터 수:	K1 * K2 * C + 1
출력 데이터:	피쳐 맵 한 장. [H, W, 1] 차원

(실제 keras에서 구현된 합성곱 계층 클래스인 Conv2D는, 선언 시 kernel_size(K1과 K2)만 지정하고, 채널 수는 직전 입력 데이터의 모양을 보고 자동으로 결정되도록 만들어져 있음.)

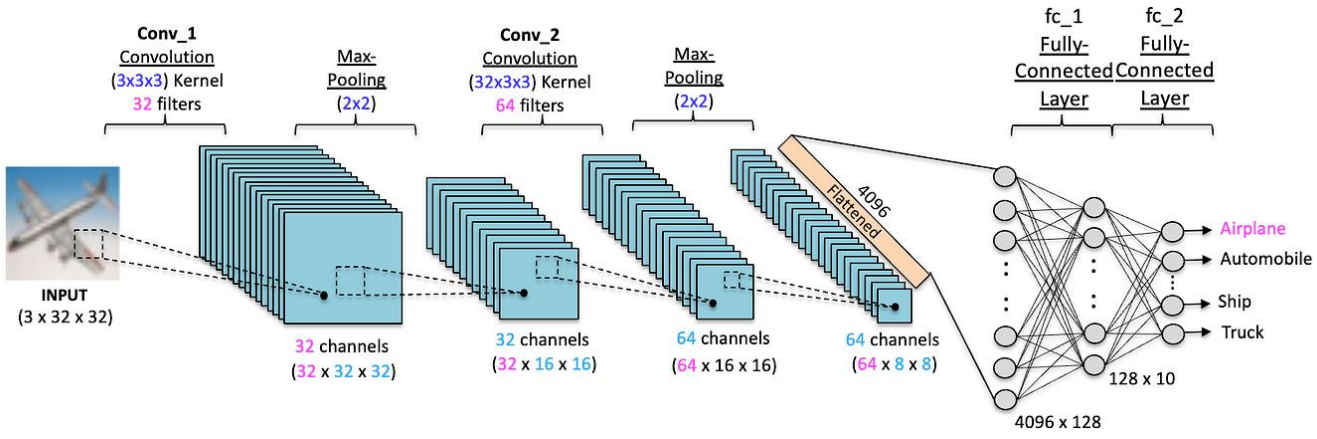
다음으로, 컬러 이미지에서도 여러 개의 필터를 사용하면, 필터 수(F) 만큼의 피쳐 맵이 나온다. 최초 합성곱 처리가 끝난 후에는, 이 F개의 피쳐 맵이 다음 단계부터 F개의 채널 역할을 하게 된다.

입력 데이터:		컬러 이미지 한 장. [H, W, C] 차원
필터 형태:		[K1, K2, C] 모양의 필터, F개가 있음
	→	
필터 파라미터 수:		(K1 * K2 * C + 1) * F
출력 데이터:		F개의 피쳐 맵. [H, W, F] 차원

컬러 이미지를 사용하는 실제 CNN 모델에서는, "컬러 이미지 한 장씩을 의미하는 샘플들의 N개가 묶음으로 들어있는" 4차원 배열을 입력으로 받는다. "실제 현실 세계의 CNN 모델"에서 사용하는 데이터 구조와 파라미터 수는 다음과 같이 확장된다.

입력 데이터:		컬러 이미지 여러 장. [N, H, W, C] 차원
필터 형태:		[K1, K2, C] 모양의 필터, F개가 있음
	→	
필터 파라미터 수:		(K1 * K2 * C + 1) * F
출력 데이터:		이미지별 F개의 피쳐 맵. [N, H, W, F] 차원

실제 CNN 구성의 예시



참고: 채널을 3번째 축이 아닌, 1번째 축에 표시한 것을 제외하고는, 앞의 설명과 동일한 구조임.
자료: https://medium.com/@lmpo/mastering-the-basics-of-convolutional-neural-networks-cnns-e98a732ea127

CNN의 파라미터 수를 비교해 보면, 합성곱 계층의 장점이 확연히 드러난다. 컬러 이미지에 대한 F개의 필터를 쓰는 합성곱 계층은, 사용하는 파라미터 수가 [(K1 * K2 * C + 1) * F]로 계산된다. 보통 필터의 K1, K2로 3 정도의 작은 수치를 쓰므로, 전체 파라미터 수는 적은 편이다.

반면에, 비슷한 신경망을 완전 연결층으로 만든다고 하면, 파라미터 수가 크게 증가한다. 컬러 이미지의 모든 정보를 노드로 본다면, 이때 노드 수는 [H*W*C]이다. 이때, 출력 노드 수를 M이라고 가정해 보자. 이 때 둘 간의 완전 연결층의 파라미터 수는 [(H*W*C)*M+M]이 된다. 이미지 사이즈의 증가에 따라, 파라미터 수가 급증하는 것을 알 수 있다.

이미지를 처리하는 AI 모델에서는, 국소 패턴 탐지 기능과 파라미터 절감 효과로 인해 CNN 사용이 일반적이다.

CNN 모델 사용 코드

keras에서는, keras.layers.Conv2D 클래스를 사용해서 2차원 합성곱 계층을 구현한다.

```
keras.layers.Conv2D(
    filters,
    kernel_size,
    …
)
```

Conv2D 계층 클래스의 필수 파라미터에는 filters와 kernel_size가 있다. filters는 필터 개수를 몇 개로 할지를 지정하는 변수다. 보통 32, 64, 128 등 2의 거듭제곱수가 사용된다. kernel_size는 앞에서 본 필터의 세로, 가로 크기다. 보통은 세로와 가로 크기를 동일하게 맞춰, 3 혹은 4 와 같은 한 개 수치로 넣는다.

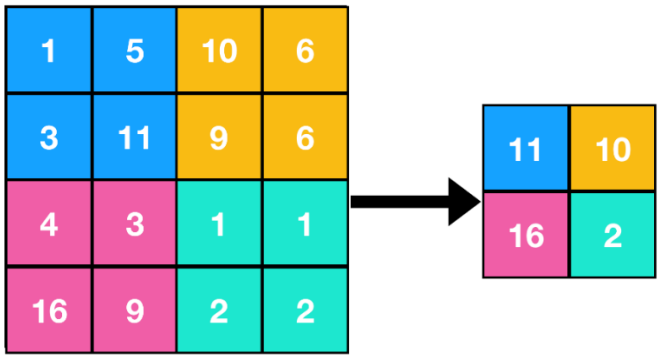
예를 들어, keras.layers.Conv2D(filters=F, kernel_size=K)로 선언했다고 하자. 그러면 keras는 Input 층의 채널 수(num_channels)를 확인해서, 크기가 [K, K, num_channels] 인 3차원 텐서 형태의 필터를 F개만큼 만들어서 CNN 계층으로 사용한다.

3) 풀링(Pooling)

CNN 구성 시, "풀링"이란 특정 맵의 공간 크기(해상도 혹은 가로 세로 크기)를 줄이면서 중요한 정보만 요약하는 연산을 말한다. 보통 MaxPooling과 AveragePooling 방식이 있다.

MaxPooling은 지정한 작은 윈도우를 가지고 전체 데이터를 격자처럼 나눈 다음, 윈도우 내에서 가장 큰 값만 남기는 연산이다(아래의 계산 예시 참고).

MaxPooling(2)의 계산 예시



자료: ResearchGate

예를 들어, [2*H, 2*W]의 입력 데이터에 MaxPooling(2)를 적용하면, 2x2 행렬마다 가장 큰 값만 남겨서, 최종 [H, W]의 출력 데이터를 만들게 된다.

AveragePooling은, 비슷한 방식으로 작은 윈도우 안 수치들의 평균값을 남기는 방식이다.

Pooling 기법은 효율적으로 정보를 축약하기 위해서 사용한다. 예를 들어, 이전의 합성곱 층에서 "사진에서 귀가 있다"라는 정보를 가지고 있다고 하면, 큰 MaxPooling 층을 통과하고 나면 "귀가 있다"란 정보가 [0, 0] 위치에 있었는지 [23, 12] 위치에 있었는지의 세세한 정보를 기억하지 않고, "귀가 있었다"란 대략적 정보만을 남기는 식이다. 이를 통해 전체 계산량도 줄이고, 효율적으로 정보를 남길 수 있다. 또한 Pooling 층은 사전에 정해진 연산만 하기 때문에, 파라미터가 없는 것도 특징이다.

2차원의 MaxPooling에 대해서, keras는 keras.layers.MaxPooling2D 클래스를 제공한다.

```
keras.layers.MaxPooling2D(
    pool_size,
    …
)
```

필수 파라미터인 pool_size는 pooling에 적용할 윈도우의 세로/가로 사이즈를 지칭한다. 보통은, 크기를 동일하게 맞춰, 2 또는 3과 같은 한 개 수치로 넣는다.

4) 금융 데이터에 대한 CNN 적용

이미지 데이터는, x축과 y축에 넓게 여러 데이터가 펴져 있는 형태다. CNN 모델은 넓은 데이터에서 유용한 패턴을 확인해, 해당 그림이 비행기인지 자동차인지 비행기가 몇 대 있는지 등을 확인할 수 있다.

금융 및 주식 데이터에서도, 데이터가 넓게 펴져 있는 형태가 비슷하다고 할 수 있다. 일자별로 생성되는 주가, 일자별로 생성되는 거래량 수치 등의 시계열 정보가 이에 해당한다. 이런 시계열 데이터에 대해서도 CNN을 사용하면 유용한 패턴을 확인해낼 수 있다.

keras가 2차원 데이터에 대한 합성곱 계층인 keras.layers.Conv2D 클래스를 지원하듯이, 1차원 데이터에 대한 합성곱 계층으로는 keras.layers.Conv1D 클래스를 지원한다. 금융 시계열 자료에서는 2차원의 가로, 세로라는 의미가 존재하지 않으므로, 이 Conv1D 계층 클래스를 쓰게 된다.

Conv2D와 Conv1D의 차이는, Conv1D는 입력 데이터에서 세로/가로 구분이 존재하지 않고, 세로의 1줄로 데이터가 구성되는 방식의 차이만 있다. (앞서 본 입력 데이터의 shape 차이로 보면, Conv2D는 전체 입력 데이터의 구조가 [N, H, W, C] 차원임. Conv1D는 전체 입력 데이터의 구조가 [N, L, C] 차원임.)

한편, 이미지 데이터의 경우에는 보통 채널에 RGB의 3개 채널이 있는 경우가 일반적이다. 금융 데이터에서는, 관찰하고자 하는 데이터 종류가 많을수록, 채널 수는 계속 늘어날 수 있다. 개별 주식의 미래 주가를 예측하기 위해, 과거 일정 기간의 일별 주가, 거래량, 시가, 고가, 저가 등의 정보를 계속 추가해서 사용하면, 채널 수는 그만큼 늘어난다.

이때, 금융 데이터에 CNN을 적용하는 경우, 필터는 어떤 의미를 가지게 되나?

예를 들어, 입력 데이터가 일별 주가 수익률 수치이고, Conv1D의 한 개 필터가 필터 사이즈(kernel_size)가 4이면서 필터 값이 [-2, 1, -1, 1]이라고 해보자. 이 경우, 주가 수익률가 필터를 통과해서 나오는 출력 맵에서는, 최근 4일 간의 주가 변동이 "큰 폭 하락 → 상승 → 하락 → 상승"인 경우에 출력값이 양수로 크게 나온다. 이는, 소위 기술적 분석에서 이야기하는 "이중 바닥" 신호와 유사하다(이중 바닥 신호가 나올 때 출력 맵의 값이 커짐). 반대로, 필터 값이 [2, -1, 1, -1]인 필터를 적용하면, 4일 간의 주가 변동이 "큰 폭 상승 → 하락 → 상승 → 하락"인 경우에 출력값이 양수로 크게 나온다. 즉, 이 필터는 "이중 천장" 신호를 감지하는 필터라고 볼 수 있다. 즉, 주가 수익률을 입력으로 받는 Conv1D 필터의 경우, 그 필터가 다양한 주가의 패턴을 감지하는 기술적 분석 지표 충족 여부를 나타내는 역할을 하는 셈이다. 만약에 입력 데이터에 일별 주가 수익률 1개 채널만 있는 것이 아니라, 시가, 고가, 저가 등 추가 정보들을 같이 넣는 멀티-채널 방식이 사용되면, 시고저종 간의 관계를 고려하는 좀더 현실적인 캔들 차트 관련 기술적 분석의 지표를 Conv1D 필터들이 흉내 낼 수 있게 된다. (여러 층의 Conv1D 층을 쌓음으로써, (이론상) 삼산형, 헤드앤숄더, 쐐기형, 엘리어트 파동 등 다양한 기술적 분석 패턴을 CNN 층이 표현해 낼 수 있음.)

앞서 말했듯이, CNN 모델에서 이런 "필터" 텐서(다차원의 배열)의 수치들은 미리 지정하는 것이 아니고, 모델의 학습 과정을 통해서 수치가 학습된다. 즉, CNN 모델에서 특정 종목의 과거 주가 시계열를 입력(feature)으로 넣고, 미래 주가 수익률을 출력(target)으로 넣고 학습을 시키면, 미래 주가 수익률을 가장 잘 설명(예측)하는 주가 패턴을 찾아서 그 패턴이 Conv1D 필터에 저장된다. 즉, 해당 종목의 미래 수익률을 잘 설명해 주는 기술적 분석 지표를 스스로 찾는 개념이라고 할 수 있다. 만약, A 종목에 잘 맞는 기술적 분석 지표와 B 종목에 잘 맞는 기술적 분석 지표가 서로 다르다면, 각각의 CNN 모델에서는 서로 다르게 학습된 필터가 만들어지게 된다.

(물론 garbage in – garbage out 원칙에 의해서, 주가 설명력이 없는 데이터만을 넣으면, 예측력이 높은 모델이 나올 수 없음.)

정리하자면, 주가 예측 시 CNN 모델을 사용하면, 입력 시계열 데이터에서 실제로 사후 주가와 연관성이 높은 데이터의 변화 패턴을 CNN 모델이 잘 잡아낼 수 있게 된다. CNN 모델은, 시계열 데이터에 대해서 단순한 MLP 모델보다 더 효율적으로 작동한다. 또한 주가 예측 CNN 모델의 입력 데이터에는, 주가와 거래량뿐만 아니라, 밸류에이션 지표, 이익 변화 지표, 주요 매크로 지표 등 유의미한 정보라면 어떤 것이라도 들어갈 수가 있다.

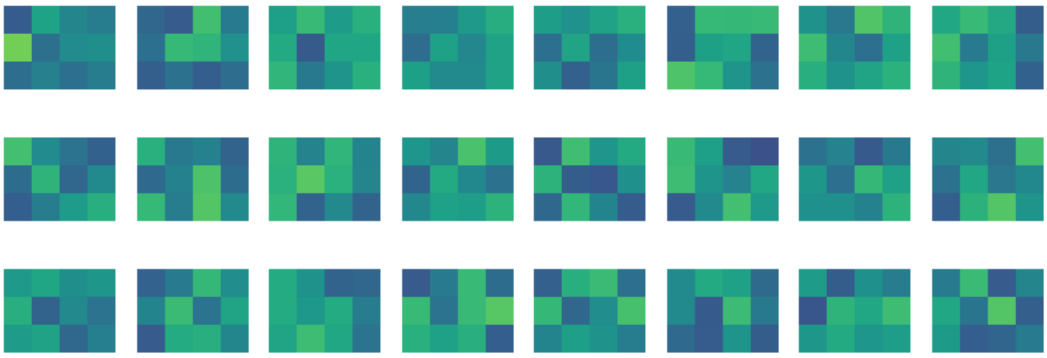
여기서 한 번, Conv1D 계층을 사용했을 때의 출력 데이터 모양과 사용 파라미터 수를 정리해 보자. 금융 시계열 정보는 가로, 세로의 2차원 배열이 아닌 1차원 배열로 구성된다. 따라서, 앞서 본 컬러 이미지 사례에서의 [N, H, W, C] 구조 대신, [H, W] → [L]로 줄어든 [N, L, C]의 데이터 구조를 가진다.

이 경우의 변화는 다음과 같다.

입력 데이터:	**시계열 데이터 여러 샘플. [N, L, C] 차원**
필터 형태:	**[K, C] 모양의 필터, F개가 있음**
→	
필터 파라미터 수:	**(K * C + 1) * F**
출력 데이터:	**데이터별 F개의 피쳐 맵. [N, L, F] 차원**

참고: 샘플 데이터 수 = N, 한 개 샘플 상의 시계열 길이 = L, 채널 수(사용 지표의 수) = C, 커널 사이즈=K, Conv1D의 필터 수 = F

Conv1D 계층의 필터 가중치 행렬 시각화 예시: 필터 수 = 32개, kernel_size = 3, 채널 수 = 4개



참고: 32개의 사각형은 각각의 개별 필터(커널)를 의미함. 한 사각형에서 세로 크기는 kernel_size만큼, 가로 크기는 채널 수만큼임. 각각의 수치 보관. 수치가 클수록 밝은 색으로 표시
자료: 삼성증권

5) CNN 모델 코딩

keras에서는 합성곱 신경망 계층으로, keras.layers.Conv1D, Conv2D, Conv3D 등의 클래스를 제공한다.

keras.layers.Conv1D: 1차원 배열 데이터. 보통 시계열 데이터 대상.

Conv2D: 2차원 데이터. 이미지 데이터 대상

Conv3D: 3차원 데이터. 동영상 혹은 의료 영상을 대상으로 사용

금융 시장 시계열 데이터에서는 주로 Conv1D 클래스를 사용한다. Conv1D의 선언 문법은 다음과 같다.

```
keras.layers.Conv1D(
    filters,
    kernel_size,
    padding="valid",
    activation=None,
    ***
)
```

Conv1D 계층 선언 시, filters 파라미터는 필터(패턴 감지기)를 몇 개 만들 것인지를 정하는 변수다. 필터 여러 개를 사용해서 다양한 패턴을 감지할 수 있다. 보통 2의 거듭제곱인 32, 64, 128 등의 수치를 많이 사용한다.

kernel_size 파라미터는 필터/커널의 관찰 윈도우(국소 패턴을 관찰하는)의 크기를 말한다. Conv2D에서 kernel_size=3은, 3x3 기반의 행렬/텐서 크기로 2차원 데이터를 관찰(합성곱 연산)하는 방식이다. Conv1D에서 kernel_size=3이면, 길이 3의 벡터의 형태로 시계열 데이터를 관찰하는 방식이다(물론 멀티 채널이라면, [3, C]의 행렬임). kernel_size 수치로는 주로 3이 가장 많이 사용된다(혹은 5, 7). 장기 시계열 상에서의 패턴을 보려면, kernel_size=3의 Conv1D 계층을 여러 층 쌓는 것을 통해서 탐지가 가능하다.

padding 파라미터에는 "valid", "same", "causal"이 가능하다. "valid"는 기본적인 합성곱을 함으로써, kernel_size가 k일 때, k-1만큼 출력 길이가 줄어든다(strides=1의 기본 상황). "same"의 경우에는, 마치 패딩 옷처럼 원 데이터의 가장자리에 0으로 채워서 출력 데이터의 길이를 원본과 동일하도록 맞춰준다. "causal"(인과형)은 시계열의 전후 관계 데이터가 있을 때, 전후 시점을 지켜주는 역할을 한다.

activation 파라미터는, keras의 다른 신경망 레이어와 동일하게, 해당 레이어에 적용되는 활성화 함수를 지정하는 부분이다(활성화 함수는 26/1/2일 자 "퀀트 모델링 A to Z - (6) MLP(Multi-Layer Perceptron) AI 투자 모델" 리포트 참고). Conv1D는 주로 은닉층으로 사용되므로, 보통 activation="relu"가 사용된다.

그 외에도 strides 파라미터(합성곱을 얼마의 간격을 띄우면서 할지를 결정) 등의 다양한 옵션이 존재한다. 또한, keras 합성곱 층 선언에서는 "채널 수"를 미리 지정할 필요가 없다. 데이터의 채널 수는, 입력 데이터에 의해서 미리 결정되는 사항이다. 따라서, 합성곱 층 선언 및 생성 시에, 입력 데이터의 채널 수를 보고 알아서 채널 수를 맞춘 상태로 필터를 생성하게 된다. 예를 들어, 입력 데이터가 채널 수=4의 형태이고([N, L, 4] 차원 데이터), Conv1D 선언 시 filters=F, kernel_size=3으로 설정해 놓으면, 실제 Conv1D 계층은 입력 데이터를 보고 난 다음 [3, 4] 차원의 필터 행렬 F개를 생성해 낸다.

뒤에서 자세히 설명하겠지만, 본 리포트의 예제 퀀트 모델에서는 직전 20일의 시계열 윈도우를 가진, 4개의 투자 정보 수치를 입력으로 사용한다(즉, 채널이 4개임). 그리고, Conv1D 계층에서 kernel_size는 3, filter 개수는 통상적인 32개로 지정했다. Conv1D 층은 은닉층으로 사용하므로, 은닉층의 일반적인 활성화 함수인 relu를 사용한다. 이 기준에서, 이번 예제에서의 Conv1D 계층 선언부 코드는 다음과 같다.

```
from tensorflow.keras import layers

layers.Conv1D(
    filters=32,
    kernel_size=3,
    padding="causal",
    activation="relu",
)
```

II. 모델 구상 및 슬라이딩 윈도우 데이터 생성

1) 모델 구상

본 리포트에서, 예시로 만들어볼 CNN 기반 퀀트 모델은 "주가, 52주 최고가, 최저가, 거래량 시계열 정보를 이용한 종목 주가 예측 모델"이다.

만약, 종목의 과거 주가, 52주 고저 가격, 거래량의 과거 시계열 패턴이 종목 미래 주가에 유의미한 영향력이 있다고 한다면(기술적 분석의 접근), 이때 CNN 모델을 사용하면 효과적인 시계열 패턴 캡처를 통해서 미래 주가 예측이 가능하다. 예시 모델에서는, 삼성전자 한 종목을 대상으로 해서 과거 주가 및 거래 정보들을 가지고 미래 주가 수익률을 예측하려고 한다. 사후에 더 실전적인 모델을 만든다고 하면, 1) 각 개별 종목마다 CNN 모델을 각각 따로 적용해 종목마다 다른 주가 결정의 패턴을 활용할 수도 있고, 2) 입력 데이터를 종목의 다양한 실적 및 투자 지표로 다변화해서 정교함을 올릴 수도 있다.

모델의 y(종속변수 혹은 target 혹은 label) 데이터는, 삼성전자의 사후 5영업일 수익률(정확히는 로그 수익률)이다. 단기적인 주가 변동을 예측하려는 목적이므로, 코스피 지수 움직임까지 고려할 필요성은 낮다. 주가 레벨은 계속 상승(발산)하기 때문에, 주가 대신 수익률 수치를 기준으로 삼는 것이 모델 안정성에서 더 낫다.

모델에서 실제 입력 데이터(X, 독립변수, feature)로 사용할 시계열 정보는, 삼성전자의 과거 1) 일별 주가 수익률, 2) 52주 고가 대비 현재가 비율, 3) 52주 저가 대비 현재가 비율, 4) 거래량의 과거 60일 이동평균 대비 비율이다. 이들 정보의 직전 20영업일 치의 시계열 데이터를, 종목의 사후 5영업일 수익률 예측에 사용한다. 기술적 분석 관점에서, 주가의 과거 시계열 패턴은 주가의 단기적인 방향성에 대한 단서를 제공할 수 있다. 단기적인 사후 5일 수익률 예측 시에는, 과거 시계열 정보를 한 달에서 세 달치 사이 정보를 참고하는 것이 무난하다고 하겠다. y 데이터에 수익률 수치를 쓴 것처럼, 과거 주가 정보도 stationary한 형태인 수익률 수치로 넣는 것이 합리적이므로, 주가 대신 수익률 형태를 입력 데이터로 썼다. 한편, 기술적 분석에서 52주 고가 대비 현재가 비율도 유용하다. 차트 기반의 분석 시 중장기적인 주가의 상승 및 하락 추세 및 지지, 저항선을 확인하기 위해서는 장기 기준의 최고가, 최저가 정보를 참고하는 것이 유용한다(HTS에서 항상 보듯이). 52주 고가 대비 현재가 비율은 일자별 [현재가 / 직전 52주 최고가]의 산식으로 계산한다. 지표 정비를 통해서, 안정적인 평균과 분산을 가진 데이터 형태로 사용할 수 있다. 52주 저가 대비 현재가 비율도 마찬가지다. 거래량 정보도, 기술적 분석에 도움이 된다. 거래량 정보는 매물대 분석 등 다양한 분석 관점을 제공할 수 있다. 여기서는 거래량의 상대적인 레벨을 확인하는 용도로, '일별 거래량의 과거 60일 이동평균 수치 대비 현재 비율'를 사용했다.

(모델의 입력 데이터)
삼성전자의 아래 정보들의 직전 20영업일 동안의 일별 수치 (t 시점에서, t-19부터 t0까지의 시계열 정보)

1. 일별 주가 로그 수익률
2. 52주 고가 대비 현재가 비율
3. 52주 저가 대비 현재가 비율
4. 일별 거래량의 직전 60일 이동평균 대비 비율

(모델의 출력 데이터)
삼성전자의 일별 사후 5영업일 주가 로그 수익률 (t 시점 기준에서, t0부터 t+5까지의 구간으로 계산)

2) 슬라이딩 윈도우 데이터 생성

일정 기간의 시계열 데이터를 입력 데이터로 사용하는 모델의 X, y 구성을 생각해 보자. 만약, 직전 20일 치 주가 수익률을 X로 사용하는 모델이라면, t0 시점의 샘플에는 [t-19 ~ t0] 구간의 주가 수익률 정보가 X에 들어가고. t+1 시점의 샘플에는 [t-18 ~ t+1] 구간의 주가 수익률 정보가 들어가게 된다. 그런데 보통 금융의 원시 시계열 데이터는 시점별로 그날의 주가 수익률이 하나 들어가는 한 줄의 데이터 형식이 된다. AI 모델에서 사용하기 위해서는, 이런 원시 데이터를 "t0 시점의 직전 20개, t+1 시점의 20개, t+2 시점의 20개 등의 X 데이터 형태"로 가공해야 한다. 이를 슬라이딩 윈도우 데이터 생성 작업이라고 한다.

2차원 이미지 데이터의 입력 데이터 구조는, 앞에서 본 것처럼 [N, H, W, C] 차원의 형태다. 이때, N는 샘플 수, H는 이미지의 픽셀 높이, W는 이미지의 픽셀 너비, C는 이미지의 채널 수(보통 RGB의 3)를 의미한다. 1차원 시계열 데이터의 입력 데이터 구조는 [N, L, C] 차원의 형태다. 이때는 N는 샘플 수, L는 시계열 길이(2차원의 H 같은), C는 채널 수를 의미한다. 원시 데이터를 가지고, 이 구조 형태의 데이터를 만들어야 한다.

우선, 모델의 X 데이터와 관련해, 데이터베이스에서 구하는 원시 데이터의 형태는 다음과 같다.

원시 데이터

date	price	52w_high	52w_low	volume
2006-01-02	13,180	13,400	8,670	20,955,550
2006-01-03	13,300	13,400	8,670	17,900,900
2006-01-04	13,980	14,000	8,670	39,290,600
...				
2025-11-26	102,800	112,400	50,800	21,314,975
2025-11-27	103,500	112,400	50,800	16,453,004
2025-11-28	100,500	112,400	50,800	15,292,277

참고: 주가, 52주 최고가, 52주 최저가는 원 단위. 거래량은 한 주 단위.

원시 데이터에는 2006/1/2부터 2025/11/28 까지의 4,912 영업일 치 일별 정보가 들어가 있다. 모델링 코드 초기에, 해당 엑셀 파일을 읽어서 Python에서 df_ts란 이름의 DataFrame 자료형으로 저장한다. 이때 dt_ts 데이터는 다음의 형태를 가진다.

```
> print(df_ts)
            price  52w_high  52w_low    volume
date
2006-01-02   13180     13400     8670  20955550
2006-01-03   13300     13400     8670  17900900
2006-01-04   13980     14000     8670  39290600
2006-01-05   13580     14000     8670  35518650
2006-01-06   13740     14000     8670  17695750
           ...       ...       ...       ...
2025-11-24   96700    112400    50800  29831172
2025-11-25   99300    112400    50800  16110054
2025-11-26  102800    112400    50800  21314975
2025-11-27  103500    112400    50800  16453004
2025-11-28  100500    112400    50800  15292277

[4912 rows x 4 columns]
```

코드에서 슬라이딩 데이터를 생성하는 전체 과정은, 별도의 함수로 구현했다. 생성 과정의 첫 번째는, 기존 원시 정보들을 가지고, 일별 주가 로그 수익률, 52주 고가 대비 현재가 비율, 52주 저가 대비 현재가 비율, 일별 거래량의 직전 60일 이동평균 대비 비율, 일별 사후 5영업일 주가 로그 수익률 등의 유용한 투자 지표 형태로 먼저 전환하는 작업이다.

```
def make_multichannel_windows(df_ts, lookback):
    """
    Parameters:
      lookback: 과거 시계열 관찰 기간
    Required columns:
      - 일자별 price(주가), 52w_high(52주 고가), 52w_low(52주 저가), volume(거래량)
    Features (channels):
      ch0: ret_1d   = 일일 로그수익률
      ch1: 52w_high_r = price / 52w_high: 52주 고가 대비 현재가 비율
      ch2: 52w_low_r  = price / 52w_low: 52주 저가 대비 현재가 비율
      ch3: vol_over_ma60 = volume / MA60(volume): 거래량의 60일 이동평균 대비 비율
    Label:
      y = log(Price[t+5]/Price[t]): 5일 후 수익률
    Returns:
      X: (N, lookback, Channel)
      y: (N,)
      idx: (N,) index of the last timestamp of each window
    """
    df_ts["ret_1d"] = np.log(df_ts.price).diff() # 1일 로그 수익률
    df_ts["52w_high_r"] = df_ts.price / df_ts["52w_high"]
    df_ts["52w_low_r"] = df_ts.price / df_ts["52w_low"]
    df_ts["vol_over_ma60"] = df_ts.volume / df_ts.volume.rolling(60).mean()
    feat_names = ["ret_1d", "52w_high_r", "52w_low_r", "vol_over_ma60"]

    # Future horizon return for label
    df_ts["y"] = np.log(df_ts.price.shift(-5) / df_ts.price) # 사후 5일의 기간 로그 수익률
```

새로 만드는 ret_1d 컬럼은, price 컬럼을 log 변환한 다음 diff() 메서드로 전일 대비차를 계산하여 일일 로그 수익률로 만들었다. 52w_high_r 컬럼은, price 컬럼을 52w_high 컬럼으로 나누어서 해당 지표를 만들었다(52w_low_r도 동일). vol_over_ma60 컬럼은, 'volume 컬럼'을 'volume 컬럼의 직전 60일간 평균값(volume.rolling(60).mean())'으로 나누어서 만들었다.

출력 데이터인 y 컬럼은, 'price 컬럼'을 'price 컬럼의 5일 후의 데이터 컬럼'으로 나누고 로그를 씌워서 사후 로그 수익률을 계산한 것이다.

해당 작업 후의 df_ts의 모습은 아래와 같다.

```
> print(df_ts)
             price  52w_high  52w_low  ...  52w_low_r  vol_over_ma60      y
date                                   ...
2006-01-02   13180     13400     8670  ...      1.520            NaN  0.043
2006-01-03   13300     13400     8670  ...      1.534            NaN  0.048
2006-01-04   13980     14000     8670  ...      1.612            NaN -0.023
2006-01-05   13580     14000     8670  ...      1.566            NaN  0.007
2006-01-06   13740     14000     8670  ...      1.585            NaN  0.010
            ...       ...      ...  ...        ...            ...    ...
2025-11-24   96700    112400    50800  ...      1.904          1.329    NaN
2025-11-25   99300    112400    50800  ...      1.955          0.717    NaN
2025-11-26  102800    112400    50800  ...      2.024          0.941    NaN
2025-11-27  103500    112400    50800  ...      2.037          0.724    NaN
2025-11-28  100500    112400    50800  ...      1.978          0.671    NaN

[4912 rows x 9 columns]

> df_ts.ret_1d.isna().sum()
1

> df_ts.vol_over_ma60.isna().sum()
59

> df_ts.y.isna().sum()
5
```

df_ts의 ret_1d 컬럼은 직전일 데이터를 참고해 일 수익률을 만들기 때문에, 주어진 2006/1/2 ~ 2025/11/28 기간 중 첫째 행인 2006/1/2의 수치는 na가 된다. vol_over_ma60 컬럼은, 직전 60일 치 데이터가 필요하므로 60번째 행인 2006/3/28부터 수치가 나오고, 그전의 59일 치는 na가 된다. y 컬럼은, 사후 5일 치 자료가 필요하므로 2025/11/21까지만 수치가 나오고, 마지막 5일 치는 na가 된다. (각각, ".isna().sum()" 코드로 na 개수 확인.)

최종 X 데이터를 준비할 때는, 한 행에 na가 하나라도 있는 행은 모두 삭제하고 정상적인 데이터만을 남겨서 진행한다. 그 코드는 다음과 같다.

```
# 시계열 자료의 앞뒤 결측치 제거
print("-" * 55)
print("First, data shape:", df_ts.shape)
df_ts = df_ts.dropna(subset=feat_names + ["y"])
print("After dropna, data shape:", df_ts.shape)
print( "-" * 55)
```

위 코드까지 실행하면, DataFrame에서 na가 있는 맨 앞의 59행과 맨 뒤의 5행이 삭제된다. 이에 따라, [4912 − 64 = 4848]에 의해, 최종적인 df_ts.shape는 [4848, 9]로 조정된다.

이제 준비된 최종 투자 지표의 DataFrame을 가지고, 각 샘플마다 "t0 시점의 직전 L일 치 시계열 자료 형태로 각각의 투자지표 데이터를 보유"하는 X 데이터를 생성해야 한다.

이 후반부 코드는 다음과 같다.

```
def make_multichannel_windows(df_ts, lookback):

    …

    feat_names = ["ret_1d", "52w_high_r", "52w_low_r", "vol_over_ma60"]

    …

    # 시계열 형태의 최종 데이터 생성
    features_ts = df_ts[feat_names].values.astype(np.float32)
    y_values = df_ts["y"].values.astype(np.float32)
    indices = df_ts.index.values

    X_list, y_list, idx_list = [], [], []

    # 시계열 데이터에서 sliding windows 생성
    for end in range(lookback - 1, len(df_ts)):
        start = end - lookback + 1
        X_list.append(features_ts[start : end + 1])  # (lookback, num_channels)
        y_list.append(y_values[end])
        idx_list.append(indices[end])

    X = np.stack(X_list, axis=0)  # (N, lookback, num_channels)
    y = np.array(y_list) # (N,)
    idx = np.array(idx_list)

    # alternative: np.lib.stride_tricks.sliding_window_view 사용
    # X = np.lib.stride_tricks.sliding_window_view(features_ts, window_shape=lookback, axis=0)
    # y = y_values[lookback - 1:]
    # idx = indices[lookback - 1:]

    return X, y, idx, feat_names
```

코드는 순차적으로, 먼저 X 데이터용 최종 사용 변수 리스트를 feat_names에 담는다. 이걸로, df_ts DataFrame에서 feat_names에 해당하는 컬럼만 뽑아서, features_ts란 numpy 배열로 저장한다. 그리고 df_ts의 y 컬럼은 y_values의 numpy 배열로 저장하고, df_ts의 날짜 인덱스는 indices 배열로 우선 저장한다.

다음 for 문에서 반복 인덱스 변수가 전체 기간을 지나가면서, features_ts를 룩백 기간만큼 자른 다음에, X_list 리스트에 계속 붙여나간다(이때, features_ts[start : end + 1] 데이터는 [lookback, 4]의 shape을 가지는 구조임). y_list, idx_list도 각 날짜(t0 시점)별로 만들어진다.

for 문이 끝난 다음에, "X = np.stack(X_list, axis=0)"를 통해서, X_list를 맨 앞에 새 축을 추가하면서 numpy 배열로 바꾸게 된다. 이 코드로 X_list란 리스트 자료가, [N, lookback, 4] 형태의 3차원 배열의 X로 변환된다.

이런 다소 번거로운 슬라이딩 윈도우 생성 작업은, np.lib.stride_tricks.sliding_window_view 통해서 자동화할 수 있다. sliding_window_view 함수는, 주어진 배열을 window_shape 만큼씩 슬라이딩 윈도우로 만드는 기능을 가진다. 이때, 슬라이딩 윈도우에 맞는 y 데이터의 정비는 "y = y[window_shape - 1 :]" 등의 코드로 만들 수 있다. y가 원본 데이터와 동일 시점일 때는 "y = y[window_shape - 1:]" 코드를, y가 1-step ahead 시점일 때는 "y = y[window_shape :]" 코드를 쓰면 된다.

여기까지가 슬라이딩 윈도우 데이터 생성 함수 코드 전체에 해당한다. 이제 이 전체 함수 코드를 가지고, 데이터 생성 작업을 직접 해보자.

```
> X, y, idx, feat_names = make_multichannel_windows(df_ts, lookback=20)


> print(idx)
['2006-04-24T00:00:00.000000000'
 '2006-04-25T00:00:00.000000000'
 ...
 '2025-11-21T00:00:00.000000000']

> print(X)
[[[ 0.0063898   0.84522206  1.3971078   0.6651367 ]
  [-0.0031898   0.8425303   1.3926585   0.67673594]
  [-0.00480385  0.8384926   1.3859844   0.6581802 ]
  ...
  [-0.00301205  0.8923284   1.4749722   0.82556134]
  [ 0.03991661  0.92866755  1.535039    1.8751091 ]
  [-0.02494627  0.90578735  1.4972191   1.4420766 ]]

 [[-0.0031898   0.8425303   1.3926585   0.67673594]
  [-0.00480385  0.8384926   1.3859844   0.6581802 ]
  [ 0.0111733   0.84791386  1.4015573   0.93477595]
  ...
  [ 0.03991661  0.92866755  1.535039    1.8751091 ]
  [-0.02494627  0.90578735  1.4972191   1.4420766 ]
  [-0.0195055   0.8882907   1.4682981   1.3012909 ]]

 ...

[[ 0.03187521  1.          2.0440881   1.1350075 ]
  [-0.02481517  0.9754902   1.9939879   1.0376037 ]
  [ 0.01000008  0.9852941   2.014028    1.0910388 ]
  ...
  [-0.01338157  0.85854095  1.8996063   0.871977  ]
  [ 0.04160925  0.8950178   1.980315    1.1585109 ]
  [-0.05938285  0.8434164   1.8661417   1.044419  ]]]

> X.shape
(4829, 20, 4)
```

실제 슬라이딩 생성 함수를 호출할 때는, lookback(시계열 길이)를 20으로 지정했다. 코드 실행 후 X 데이터를 print하면, 위의 화면처럼 나온다. print(X) 출력 결과에서, 맨 처음에 [0.0063898 0.84522206 1.3971078 0.6651367]의 4개 항목이 나오는데, 이는 X[0, 0, :]의 데이터로 "첫 번째 샘플의 시계열 상 첫째일(19일전)의 4개 지표(일일 로그 수익률 등 4개 정보)"를 의미한다. 그다음 [-0.0031898 0.8425303 1.3926585 0.67673594]는 X[0, 1, :]의 4개 수치다. 이렇게 X[0, 19, :]까지 20번의 데이터가 나오면서, X[0, :, :]의 데이터를 구성한다.

이런 세트로 총 4,829개 샘플이 모여서 총 [4829, 20, 4] 차원의 X 데이터를 구성한다.

(슬라이딩 직전의 전체 데이터 수는 4,848개였다. 20일 치를 슬라이딩하므로, 20일치 가 없는 초반 19개 샘플이 줄어든다. 이에 따라 최종 샘플 수는 4,848 – 19 = 4,829개가 된다.)

처음에 출력한 idx 배열에는 "슬라이딩 윈도우 상의 마지막 시점 날짜"가 들어간다(직전 함수코드 안에서, idx_list.append(indices[end])의 코드를 사용함). 출력 결과를 보면, idx[0]에는 "2006-04-24" 날짜가 들어있다. 이는 X 데이터의 0번 샘플(X[0])이 "2006-04-24" 날짜 데이터임을 말한다. X[0] 데이터는 2006/4/24 기준으로 직전 20일 치인 "2006-03-28 ~ 2006-04-24" 구간의 시계열 정보를 담고 있다.

이것으로, 신경망 모델에 사용할 슬라이딩 윈도우 시계열 형태의 X 데이터 등을 모두 준비했다. CNN용 X, y 생성 함수 전체 코드는 다음과 같다.

CNN용 X, y 생성 함수 전체 코드

```
# ------------------------------------------------------------
# CNN용 X, y 생성 함수
# 멀티채널 시계열 특징 + 라벨 생성 + (lookback) 윈도우로 X,y 만들기
#    X shape: (num_samples, lookback, num_channels)
#    y shape: (num_samples,) 미래 5일 기준 수익률
# ------------------------------------------------------------
def make_multichannel_windows(df_ts, lookback):
    """
    Parameters:
      lookback: 과거 시계열 관찰 기간
    Required columns:
      - 일자별 price(주가), 52w_high(52주 고가), 52w_low(52주 저가), volume(거래량)
    Features (channels):
      ch0: ret_1d   = 일일 로그수익률
      ch1: 52w_high_r = price / 52w_high: 52주 고가 대비 현재가 비율
      ch2: 52w_low_r  = price / 52w_low: 52주 저가 대비 현재가 비율
      ch3: vol_over_ma60 = volume / MA60(volume): 거래량의 60일 이동평균 대비 비율
    Label:
      y = log(Price[t+5]/Price[t]): 5일 후 수익률
    Returns:
      X: (N, lookback, Channel)
      y: (N,)
      idx: (N,) index of the last timestamp of each window
    """
    df_ts["ret_1d"] = np.log(df_ts.price).diff() # 1일 로그 수익률
    df_ts["52w_high_r"] = df_ts.price / df_ts["52w_high"]
    df_ts["52w_low_r"] = df_ts.price / df_ts["52w_low"]
    df_ts["vol_over_ma60"] = df_ts.volume / df_ts.volume.rolling(60).mean()
    feat_names = ["ret_1d", "52w_high_r", "52w_low_r", "vol_over_ma60"]

    # Future horizon return for label
    df_ts["y"] = np.log(df_ts.price.shift(-5) / df_ts.price) # 사후 5일의 기간 로그 수익률

    # 시계열 자료의 앞뒤 결측치 제거
    print("-" * 55)
    print("First, data shape:", df_ts.shape)
    df_ts = df_ts.dropna(subset=feat_names + ["y"])
    print("After dropna, data shape:", df_ts.shape)
    print( "-" * 55)
```

```
    # 시계열 형태의 최종 데이터 생성
    features_ts = df_ts[feat_names].values.astype(np.float32)
    y_values = df_ts["y"].values.astype(np.float32)
    indices = df_ts.index.values

    X_list, y_list, idx_list = [], [], []

    # 시계열 데이터에서 sliding windows 생성
    for end in range(lookback - 1, len(df_ts)):
        start = end - lookback + 1
        X_list.append(features_ts[start : end + 1])  # (lookback, num_channels)
        y_list.append(y_values[end])
        idx_list.append(indices[end])

    X = np.stack(X_list, axis=0)  # (N, lookback, num_channels)
    y = np.array(y_list) # (N,)
    idx = np.array(idx_list)

    # alternative: np.lib.stride_tricks.sliding_window_view 사용
    # X = np.lib.stride_tricks.sliding_window_view(features_ts, window_shape=lookback, axis=0)
    # y = y_values[lookback - 1:]
    # idx = indices[lookback - 1:]

    return X, y, idx, feat_names
```

III. 신경망 모델 선언, 학습, 평가

1) 신경망 모델 선언

이제, CNN 모델을 선언 및 컴파일하는 함수 코드를 작성하고자 한다. 우선, Conv1D 계층과 MaxPooling1D 계층에 대한 설명 및 사용 코드는 앞에서 정리하였다.

여기서, 구상하는 CNN 모델의 전체 구조는 다음과 같다.

CNN 모델의 전체 네트워크 구성도

0. 입력 데이터: 4개 투자 지표의 과거 20일치 시계열 데이터.
(4개 지표는 ret_1d, 52w_high_r, 52w_low_r, vol_over_ma60 임. Shape=[N, 20, 4]) →

1. Conv1D 계층 (필터 수 32개) →

2. MaxPooling1D 계층 (pool_size=5) →

3. Flatten 계층 →

4. Dropout 층 →

5. Dense 계층 →

6. Dropout 계층 →

7. Output(Dense) 계층: 삼성전자의 사후 수익률 데이터. 데이터 shape는 [N,]임

여기에 나오는, Flatten 계층은, 컬러 이미지 같은 (다채널의) 3차원 데이터나 멀티-채널 시계열 같은 2차원 데이터를, 수치를 유지한 채 1차원으로 "펴는" 레이어를 말한다. 다차원의 데이터를 일반적인 Dense 층과 연결하기 위해서 기계적으로 삽입하는 계층이다.

전체 코드상에서 신경망 모델 선언 함수 부분은, 유연한 대응이 가능하도록 lookback(시계열 길이), num_channels(채널 수)를 별도 입력 파라미터로 받도록 구성했다.

실제 작성 코드는 다음과 같다.

```
# ------------------------------------------------------------
# 멀티채널 Conv1D 모델 (kernel_size=3 커널) 정의 함수
# ------------------------------------------------------------
def build_conv1d_model(lookback: int, num_channels: int):

    model = keras.Sequential([
        layers.Input(shape=(lookback, num_channels)),
        # Conv1D 레이어: 과거 시점의 정보를 활용할 수 있도록 "causal" 패딩 사용
        layers.Conv1D(
            filters=32,
            kernel_size=3,
            padding="causal",
            activation="relu",
        ),
        layers.MaxPooling1D(5),
        layers.Flatten(),
        layers.Dropout(0.2),
        layers.Dense(32, activation="relu", name="hidden"),
        layers.Dropout(0.2),
        layers.Dense(1, activation=None, name='output'),  # 회귀 모델을 위해 선형 활성화 (None)
    ])

    model.compile(optimizer=keras.optimizers.Adam(learning_rate=1e-3), loss="mse", metrics=["mae"])

    return model


# model = build_conv1d_model(lookback=X.shape[1], num_channels=X.shape[2])
```

코드를 보면, Keras.Sequential 안에서 layers.Input는 "shape=(lookback, num_channels))"를 사용해, (1개 샘플 기준의) 샘플 데이터 형태를 2차원의 배열로 지정한다.

layers.Conv1D는, '커널 사이즈를 3으로 하는 필터 32개를 생성하는' Conv1D 계층을 만든다.

layers.MaxPooling1D(5)는, 이전 Conv1D의 32개 필터 맵상에서 각 5개 시계열 항목마다 최댓값을 뽑아서 저장해 데이터를 압축하는 역할을 한다. pool_size=5를 쓴 것은, 각 5영업일 간의 이벤트 발생 여부를 기록하는 의미에 해당한다. 예를 들어, 만약 직전 한 개 필터 맵이 "이중 바닥"의 출현 여부를 감지하는 필터라고 하면, MaxPooling1D(5)를 통한 다음에는, 직전 4주의 기간 동안 각 주마다 "이중 바닥" 패턴이 출현했었는지를 기록하는 방식으로 정보를 압축한다. pool_size를 다르게 변경하면, 예를 들어 이틀 간격 혹은 10일 간격의 출현 여부를 저장하는 식으로 변경할 수 있다. 시계열 전체 구간 내에서 출현 여부가 한 번이라도 있었는지만을 기록하는 방식이 필요하면, MaxPooling1D 층이 아닌 GlobalMaxPooling1D 층을 사용해 크게 압축하는 방법도 있다.

다음의 Flatten 계층은, 직전의 [시계열 길이, 채널 수] 형태의 2차원 데이터를 평평하게 펴서 1차원 Dense 층과 연결이 가능하게 만들어 주는 역할을 한다.

그다음은 Dropout → Dense → Dropout → Output(Dense) 층이 평이하게 연결된다. 마지막 Output 층에는, 나중에 삼성전자의 사후 수익률 수치가 들어갈 예정이다. 따라서, 노드 수가 1개인 Dense 층으로 선언했다.

마지막의 model.compile 코드에서는, 모델 사용에 필요한 optimizer, loss, metrics의 지정 정보가 들어간다. 이와 관련한 상세 내용도 26/1/2일 자 "퀀트 모델링 A to Z - (6) MLP(Multi-Layer Perceptron) AI 투자 모델" 리포트에 찾아볼 수 있다.

신경망 모델을 실제 생성하고 나면, 각 계층별 데이터의 형태는 model.summary() 코드를 통해서 확인할 수 있다.

lookback=20, channels=4로 실제 신경망 모델 선언 후, summary 결과를 확인하면 다음과 같다.

```
> build_conv1d_model(lookback=20, num_channels=4)
<keras.src.engine.sequential.Sequential at 0x19891257710>


> model.input.shape
TensorShape([None, 20, 4])


> model.summary()
Model: "sequential_8"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv1d_8 (Conv1D)           (None, 20, 32)            416

 max_pooling1d_8 (MaxPoolin  (None, 4, 32)             0
 g1D)

 flatten_8 (Flatten)         (None, 128)               0

 dropout_16 (Dropout)        (None, 128)               0

 hidden (Dense)              (None, 32)                4128

 dropout_17 (Dropout)        (None, 32)                0

 output (Dense)              (None, 1)                 33

=================================================================
Total params: 4577 (17.88 KB)
Trainable params: 4577 (17.88 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

model.summary()코드를 통해서, 각 신경망 층의 데이터 구조를 확인할 수 있다. 다만, summary()만으로는 입력 레이어의 정보를 볼 수 없는 경우가 대부분이다(입력 레이어는 신경망 그래프의 시작점이라서, 보통 summary() 출력에서 빠짐). 입력 데이터의 구조는, 별도의 "model.input.shape" 코드로 확인이 가능하다.

이번 에시에서는 layers.Input(shape=(20, 4))로 선언했다. 따라서, model.input.shape 결과는 [None, 20, 4]로 나온다. 배열의 첫 번째 축은 샘플 번호를 기록하는 곳이므로, 항상 None으로 표시된다.

(모델에서는 항상 다양한 샘플 수의 데이터를 처리할 수 있음. 여기서 실제로는 4,829개 샘플을 쓰는 상황임.)

다음으로, model.summary() 결과를 살펴보자.

1) Conv1D 처리:

입력 데이터는 [None, 20, 4]인 상태다. Conv1D 선언 시에 kernel_size=3, filers=32로 선언했다. 입력 데이터의 채널 수가 4이기 때문에, 이에 따라 합성곱 필터는 [3, 4] 형태의 필터가 32개 생겨난다. 해당 필터 한 개를 적용하면 한 개 특성 맵은 [None, 20] 차원이다. 이 특성 맵이 32개 있으므로, Conv1D 처리 후 데이터 구조는 [None, 20, 32]가 된다.

또한 이때 필터의 총 파라미터 수는 [K * C + 1] * F = (3 * 4 + 1) * 32 = 416개가 된다(Parma # 부분의 416 표시).

입력 데이터:	**시계열 데이터 여러 샘플. [N, L, C] 차원**
필터 형태:	**[K, C] 모양의 필터, F개가 있음**
→	
필터 파라미터 수:	**(K * C + 1) * F**
출력 데이터:	**데이터별 F개의 피쳐 맵. [N, L, F] 차원**

참고: 샘플 데이터 수 = N, 한 개 샘플 상의 시계열 길이 = L, 채널 수(사용 지표의 수) = C, 커널 사이즈=K, Conv1D의 필터 수 = F

2) MaxPooling1D(5) 처리:

MaxPooling1D(5)는 시계열 데이터 상 5개의 최댓값만 남겨 1/5로 데이터를 줄인다. 따라서 데이터 구조는 [None, 20, 32]에서 [None, 4, 32]로 바뀐다. MaxPooling1D는 정해진 기계적 처리만 할 뿐이며, 학습 파라미터가 없다. 따라서, 파라미터 수는 0이다.

3) Flatten 처리:

Flatten은 2차원 데이터를 1차원으로 펴는 역할이다. 데이터 구조만 [None, 4, 32]에서 [None, 128]로 펴진다. 역시나, 학습 파라미터는 없다.

4) Dropout 층: 이 층은 dropout 기능의 구현용이다("(6) MLP(Multi-Layer Perceptron) AI 투자 모델" 리포트 참고). 데이터 구조는 [None, 128] 그대로 유지다. 파라미터는 없다.

5) hidden 층:

layers.Dense(32) 코드를 통해, 32개 노드를 가진 층으로 정의했다. 데이터 구조는 [None, 32]으로 축소된다. 이 완전 연결층에 필요한 파라미터 수는 128*32 + 32 = 4,128개다. (직전 층 128개 노드와 현재 층 32개 노드 간의 완전 연결 가중치와 32개 노드의 절편 정보의 수.)

6) Dropout 층: 이 층은 dropout 기능의 구현용이다. 데이터 구조는 [None, 128] 그대로 유지다. 파라미터는 없다.

7) Output 층:

layers.Dense(1) 코드를 통해, (사후 수익률) 1개 노드를 가진 층으로 정의했다. 데이터 구조는 [None, 1]로 축소된다. 이 완전 연결층에 필요한 파라미터 수는 32*1 + 1 = 33개다.

이 구조에 따라서, 신경망 모델의 전체 파라미터 수는 416 + 4,128 + 33 = 4,577개로 결정된다.

이상으로 CNN 모델의 모델 생성 코드가 완성되었다.

2) 모델 학습 및 평가

여기서는, 실제 데이터를 불러오고 신경망 모델을 만든 다음, 실제 모델 학습을 시키고 그의 평가 결과를 확인하는 코드를 작성한다.

이 부분은, 이전 "(6) MLP(Multi-Layer Perceptron) AI 투자 모델" 리포트에서의 구현과 상당히 흡사하다. 참고로, 금융 시계열을 사용하므로, TimeSeriesSplit의 시계열 교차 검증법을 여기에서도 사용한다.

메인 코드에서의, 모델 학습 및 평가 코드는 다음과 같다.

```
# ------------------------------------------------------------
# MAIN
# ------------------------------------------------------------
if __name__ == "__main__":

    # ------------------------------
    # 0. Raw 데이터 로딩
    # ------------------------------
    df_ts = pd.read_excel('timeseries.xlsx', sheet_name='data', skiprows=13, index_col=0)


    # ------------------------------
    # 1. CNN용 X, y 생성
    # ------------------------------
    X, y, idx, feat_names = make_multichannel_windows(df_ts, lookback=20)

    print("Feature channels:", feat_names)
    print("X.shape:", X.shape)
    print("y.shape:", y.shape, "\n" + "-" * 55)

    # X의 각 채널별 통계량 확인
    # X (samples, lookback, channels) -> (samples * lookback, channels) 2D로 변환
    print("\n--- X, channels description ----------------------------")
    print(pd.DataFrame(X.reshape(-1, X.shape[-1]), columns=feat_names).describe().iloc[1:3])
    print("-" * 55)


    # ------------------------------
    # 2. CNN 모델 학습 및 평가
    # ------------------------------
    tscv = TimeSeriesSplit(n_splits=5, gap=5)

    all_val_losses = []
    all_val_maes = []

    # ------------------------------
    # Time Series Cross Validation (Expanding)
    # ------------------------------
    for fold, (train_idx, val_idx) in enumerate(tscv.split(X), start=1):
        print(f"\n===== Fold {fold}/5 =====")
        print(f"Train index: {train_idx[0]} ~ {train_idx[-1]}")
        print(f"Valid index: {val_idx[0]} ~ {val_idx[-1]}")

        X_train_raw, X_val_raw = X[train_idx], X[val_idx]
        y_train, y_val = y[train_idx], y[val_idx]

        # 데이터 표준화: train 데이터에만 fit하고, train/val 모두 transform
        scaler = StandardScaler()
```

```
        ## Reshape, fit, transform, and reshape back for train set
        X_tr_shape = X_train_raw.shape
        X_train = scaler.fit_transform(X_train_raw.reshape(-1, X_tr_shape[-1])).reshape(X_tr_shape)
        ## Reshape, transform, and reshape back for val, test set
        X_val_shape = X_val_raw.shape
        X_val = scaler.transform(X_val_raw.reshape(-1, X_val_raw.shape[-1])).reshape(X_val_shape)
        ## describe
        print("--- X_train (After scaling), channels description -----")
        print(pd.DataFrame(X_train.reshape(-1, X_tr_shape[-1]), columns=feat_names).describe()[:3])


        # 매 fold마다 모델 새로 생성 (가중치 초기화)
        model = build_conv1d_model(lookback=X.shape[1], num_channels=X.shape[2])

        # 콜백 정의: 조기 종료 및 학습률 스케줄러
        early_stopping_cb = EarlyStopping(patience=10, restore_best_weights=True)
        reduce_lr_cb = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=5, min_lr=0.00001)
        callbacks = [early_stopping_cb, reduce_lr_cb]

        # 학습 (시계열이므로 shuffle=False)
        history = model.fit(
            X_train, y_train,
            validation_data=(X_val, y_val),
            epochs=200,
            callbacks=callbacks,
            shuffle=False,
            verbose=0,
        )

        # 해당 fold 검증 성능 기록
        val_loss, val_mae = model.evaluate(X_val, y_val, verbose=0)
        print(f"Fold {fold} - val_loss: {val_loss:.4f}, val_mae: {val_mae:.4f}")
        all_val_losses.append(val_loss)
        all_val_maes.append(val_mae)


# ------------------------------
# 마지막 Fold의 학습 과정 시각화
# ------------------------------
plt.plot(history.history['loss'], label='loss')
plt.plot(history.history['val_loss'], label='val_loss')
plt.title('Model Loss (Last Fold)')
plt.legend()
plt.show()

# ------------------------------
# 전체 fold 성능 요약
# ------------------------------
print("\n===== Cross-Validation Result (Expanding Window) =====")
print(f"val_loss (per fold): {[f'{v:.4f}' for v in all_val_losses]}")
print(f"val_mae  (per fold): {[f'{v:.4f}' for v in all_val_maes]}")
print(f"mean val_loss: {np.mean(all_val_losses):.4f}")
print(f"mean val_mae : {np.mean(all_val_maes):.4f}")
```

메인 실행 코드의 전반부는 "0. Raw 데이터 로딩", "1. CNN용 X, y 생성" 파트까지다. pd.read_excel로 엑셀 데이터를 읽고, 앞서 만든 슬라이딩 생성 함수를 써서 CNN용 X와 y 데이터를 준비해 놓는 것까지 담당한다.

메인의 후반부는, Time Series Cross Validation의 for 문 안에서 test와 valid 데이터를 매번 준비한 다음, 모델 선언을 하고, history = model.fit()을 통해서, 모델 학습 및 평가 기록을 하고, 이를 누적하는 단계로 되어 있다. 후반부 루틴의 구성은 이전 "(6) MLP(Multi-Layer Perceptron) AI 투자 모델" 리포트와 거의 유사하다.

TimeSeriesSplit 사용에 있어서는, y 데이터 자체가 사후 5일 기준 수익률을 쓰므로, 인접한 y 데이터 간의 데이터 유출을 막기 위해서 "gap=5" 옵션을 사용했다.

(model.fit, callbacks, model.evaluate, history.history 등의 전체적인 내용은, 이전 리포트 참고.)

간단히 언급하면, 모델 학습(model.fit) 코드에서는 1) X_train, y_train을 가지고 학습하며, 2) 전체 샘플의 학습 횟수(epoch)는 200회로 하고, 3) 조기 중단 및 학습률 조정의 callback을 사용하며, 4) 학습 완료 후에는 X_val, y_val을 가지고 검증 결과를 확인하는 과정을 진행한다.

이번 코드에서 특징적인 부분은, "데이터 표준화" 처리 부분이다.

딥러닝/머신러닝에서 데이터 표준화 기능을 가진 sklearn.preprocessing.StandardScaler는, 보통 1개 샘플이 벡터로 표시되는 [n_samples, n_features] 구조의 2차원 배열 데이터에만 쓸 수 있다. 그런데, 이번 멀티 채널 CNN 모델의 X 데이터 구조는 [N, 20, 4]의 3차원 배열이다. 따라서, 바로 StandardScaler를 사용할 수 없다.

이때 [N, 20, 4]의 X 데이터는 사실 ,4개의 서로 다른 투자 지표의 20일 치 시계열의 뭉치를 의미한다. 따라서 4개 그룹으로만 나누면, 그 안에서는 동질한 데이터에 해당한다. 따라서, 이때 표준화 작업은 1) [N, 20, 4]의 3차원 배열 데이터를 [N*20, 4]의 2차 배열로 전환하고, 2) 이를 표준화하고, 3) 그 결과를 다시 3차원 배열 형태로 변환하는 절차를 사용한다.

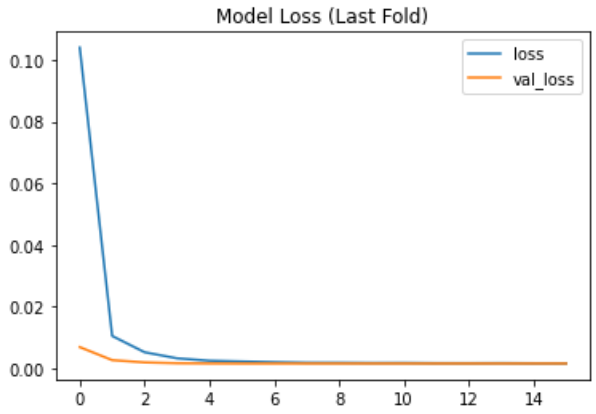
scaler.fit_transform이 있는 구문을 보자. X_train_raw.shape이 [N, 20, 4] 이므로, "X_train_raw.reshape(-1, X_tr_shape[-1])" 코드는, X_train_raw의 shape를 [-1, 4]로 재조정하는 작업을 한다. 즉, X_train_raw를 [N*20, 4]의 데이터 구조로 바꾼다. 이 바뀐 데이터에 StandardScaler가 적용된다. 그후에는 다시 원래의 [N, 20, 4]로 데이터 형태를 되돌린다. 이 코드 구문이, CNN 모델 데이터의 표준화(standardization) 처리 코드로 많이 사용된다.

IV. 모델 실행 결과 및 정리

이상의 전체 코드를 실행하면, CNN 모델링의 출력 결과물이 다음의 형태로 나온다.

```
# 실행 결과

Feature channels: ['ret_1d', '52w_high_r', '52w_low_r', 'vol_over_ma60']
X.shape: (4829, 20, 4)
y.shape: (4829,)
------------------------------------------------------

--- X, channels description ---------------------------
      ret_1d  52w_high_r  52w_low_r  vol_over_ma60
mean   0.000       0.861      1.315          1.007
std    0.018       0.097      0.245          0.408
------------------------------------------------------

===== Fold 1/5 =====
Train index: 0 ~ 803
Valid index: 809 ~ 1612
--- X_train (After scaling), channels description -----
         ret_1d  52w_high_r  52w_low_r  vol_over_ma60
count 16080.000   16080.000  16080.000      16080.000
mean     -0.000       0.000      0.000          0.000
std       1.000       1.000      1.000          1.000
Fold 1 - val_loss: 0.0016, val_mae: 0.0315

===== Fold 2/5 =====
Train index: 0 ~ 1607
Valid index: 1613 ~ 2416
--- X_train (After scaling), channels description -----
         ret_1d  52w_high_r  52w_low_r  vol_over_ma60
count 32160.000   32160.000  32160.000      32160.000
mean      0.000       0.000     -0.000          0.000
std       1.000       1.000      1.000          1.000
Fold 2 - val_loss: 0.0011, val_mae: 0.0262

===== Fold 3/5 =====
Train index: 0 ~ 2411
Valid index: 2417 ~ 3220
--- X_train (After scaling), channels description -----
         ret_1d  52w_high_r  52w_low_r  vol_over_ma60
count 48240.000   48240.000  48240.000      48240.000
mean     -0.000       0.000      0.000         -0.000
std       1.000       1.000      1.000          1.000
Fold 3 - val_loss: 0.0012, val_mae: 0.0284

===== Fold 4/5 =====
Train index: 0 ~ 3215
Valid index: 3221 ~ 4024
--- X_train (After scaling), channels description -----
         ret_1d  52w_high_r  52w_low_r  vol_over_ma60
count 64320.000   64320.000  64320.000      64320.000
mean      0.000       0.000      0.000         -0.000
std       1.000       1.000      1.000          1.000
Fold 4 - val_loss: 0.0014, val_mae: 0.0274

===== Fold 5/5 =====
Train index: 0 ~ 4019
Valid index: 4025 ~ 4828
```

```
--- X_train (After scaling), channels description -----
         ret_1d  52w_high_r  52w_low_r  vol_over_ma60
count 80400.000   80400.000  80400.000      80400.000
mean      0.000       0.000      0.000         -0.000
std       1.000       1.000      1.000          1.000
Fold 5 - val_loss: 0.0015, val_mae: 0.0302
```




```
===== Cross-Validation Result (Expanding Window) =====
val_loss (per fold): ['0.0016', '0.0011', '0.0012', '0.0014', '0.0015']
val_mae  (per fold): ['0.0315', '0.0262', '0.0284', '0.0274', '0.0302']
mean val_loss: 0.0014
mean val_mae : 0.0288
```

첫 번째로는, make_multichannel_windows 함수 실행 후에, 피처 이름 및 X.shape, y.shape을 확인한 결과가 나온다. X.shape: (4829, 20, 4), y.shape: (4829,)로 나온다.

다음에는, X 데이터의 채널별 통계량을 보여준다. X 데이터는 4개 채널의 20일 치 시계열 자료를 샘플별로 보관하는 [4829, 20, 4] 형태의 자료다. 따라서, 채널별 평균 및 표준편차를 보려면, 채널 기준으로 데이터를 펴서 describe() 작업을 진행하면 된다. 이때, "X.reshape(-1, X.shape[-1])" 코드를 통해서, [4829, 20, 4] 형태 자료를 [96580(=4829*20), 4] 형태의 자료로 임시로 변환한다. 채널별 통계량을 보면, 일일 로그수익률(ret_1d)는 평균이 0.000이다. 52주 고가 대비 현재가 비율(52w_high_r)는 평균이 0.861이다(직전 1년의 고점 대비 현재 주가는 항상 1배 이하로 나와야 함). 52주 저가 대비 현재가 비율(52w_low_r)는 평균이 1.315다. 일일 거래량의 60일 이동평균 대비 비율(vol_over_ma60)은 평균이 1.007다.

다음으로, TimeSeriesSplit 시계열 교차 검증의 출력 결과를 보자.

첫 번째 교차 검증에서는, 사용할 데이터로 train_index=0~803, vaild_index=809~1612가 사용되었다. 실제 날짜로 보면, 2006/4/24 ~ 2009/7/23 기간의 데이터를 가지고 학습한 뒤, 2009/7/31~2012/10/17 기간의 검증 데이터에 대해서 모델 성능을 검증한다. 첫 번째 폴드에서 CNN 모델이 삼성전자 사후 수익률을 예측하는 성능을 검증하면, 평균 절대 오차가 3.15%로 나온다(val_mae로 확인).

두 번째 교차 검증에서는, 데이터로 train_index=0~803, vaild_index=809~1612가 사용되었다. 즉, Expanding Window 형태로 학습 구간을 늘렸다. 두 번째 폴드에서 모델의 검증 성능은 평균 절대 오차 2.62%로 나왔다(val_mae). 이런 식으로 5개의 교차 검증 결과가 각각 출력된다.

그다음의 그래프 부분은, 마지막 폴드의 history 정보를 사용해, 마지막 폴드 상에서의 각 epoch 진행에 따른 모델의 훈련 및 검증 평가 결과를 시각화한 것이다. 그래프상의 loss 선을 보면, epoch가 진행될 동안, train 데이터 기준의 손실값(MSE로 지정)이 가파르게 떨어진다. val_loss 선은 완만하게 하락하다 정체되는 것을 볼 수 있다. 그래프상 X축(epoch 시점)이 15에서 끊겨 있으므로, 조기 중단 callback 발동으로 인해 모델 학습이 빨리 중단된 것을 알 수 있다.

마지막 "Cross-Validation Result" 부분은, 교차 검증 결과를 모두 누적하여 보여주는 결과다. 5번의 교차 검증에서. 검증 평가 지표인 val_mae(검증 데이터의 평균 절대 오차)는 평균적으로 2.88%의 수치를 보였다.

이상을 보면, 이번 예시의 CNN 모델이, 실제 모델 예측력이 아주 우수하다고 보기는 어렵다. 현재는, CNN의 설명을 위해서 간단한 형태의 데이터와 모델 구조를 사용한 측면이 크다. 실제로 개별 종목의 미래 주가 변동은, 과거 주가 및 거래량 등의 거래 정보만으로 100% 예측하는 것이 불가능하다고 하겠다. 종목의 실적 전망 변화, 적용 밸류에이션의 변화, 소속 시장 움직임, 미국 증시의 방향성 등 실제 고려 요소는 훨씬 다양한 편이다. 따라서, 실전 모델에서는 훨씬 다양한 영향 변수의 추가 및 CNN 모델 구조의 고도화 등이 더 필요하다고 판단한다.

정리

퀀트 투자와 관련된 입력 및 출력 데이터를 가지고, CNN 투자 모델을 구축하고 및 평가하는 단계는 여기서 끝난다.

이후의 실전 투자 모델 구축 시에는, 실제 투자 환경상의 백테스팅 과정이 필요하다. 시간순으로 진행하면서, 각 시점별로 1) CNN 모델을 정의하고, 2) 가용 데이터로 모델 학습 및 하이퍼 파라미터 튜닝을 진행하고, 3) 예측 결과로 투자를 집행하여, 투자 시뮬레이션에 대한 전통적 투자 성과 평가를 통해서 투자 전략에 대해 검증해 보는 과정이 필요하다.

정리하면, CNN 모델은 시계열에서의 패턴 감지라는 장점으로 인해, 금융 시계열 예측 시에 활용 가능성이 높다. 또한 최근에는 LSTM 등 재귀 형태 모델과 CNN을 결합하는 방식 등의 다양한 딥러닝 모델 접근법이 개발되고 있다. 따라서, CNN 모델의 구조와 사용법을 숙지할 필요가 있다고 판단한다.

V. Appendix: CNN 퀀트 모델의 전체 코드

```
# 전체 코드

# -*- coding: utf-8-sig -*-
#%%

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import TimeSeriesSplit
from sklearn.preprocessing import StandardScaler
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau

# ------------------------------------------------------------
# CNN용 X, y 생성 함수
# 멀티채널 시계열 특징 + 라벨 생성 + (lookback) 윈도우로 X,y 만들기
#    X shape: (num_samples, lookback, num_channels)
#    y shape: (num_samples,) 미래 5일 기준 수익률
# ------------------------------------------------------------
def make_multichannel_windows(df_ts, lookback):
    """
    Parameters:
      lookback: 과거 시계열 관찰 기간
    Required columns:
      - 일자별 price(주가), 52w_high(52주 고가), 52w_low(52주 저가), volume(거래량)
    Features (channels):
      ch0: ret_1d   = 일일 로그수익률
      ch1: 52w_high_r = price / 52w_high: 52주 고가 대비 현재가 비율
      ch2: 52w_low_r  = price / 52w_low: 52주 저가 대비 현재가 비율
      ch3: vol_over_ma60 = volume / MA60(volume): 거래량의 60일 이동평균 대비 비율
    Label:
      y = log(Price[t+5]/Price[t]): 5일 후 수익률
    Returns:
      X: (N, lookback, Channel)
      y: (N,)
      idx: (N,) index of the last timestamp of each window
    """
    df_ts["ret_1d"] = np.log(df_ts.price).diff() # 1일 로그 수익률
    df_ts["52w_high_r"] = df_ts.price / df_ts["52w_high"]
    df_ts["52w_low_r"] = df_ts.price / df_ts["52w_low"]
    df_ts["vol_over_ma60"] = df_ts.volume / df_ts.volume.rolling(60).mean()
    feat_names = ["ret_1d", "52w_high_r", "52w_low_r", "vol_over_ma60"]

    # Future horizon return for label
    df_ts["y"] = np.log(df_ts.price.shift(-5) / df_ts.price) # 사후 5일의 기간 로그 수익률

    # 시계열 자료의 앞뒤 결측치 제거
    print("-" * 55)
    print("First, data shape:", df_ts.shape)
    df_ts = df_ts.dropna(subset=feat_names + ["y"])
    print("After dropna, data shape:", df_ts.shape)
    print( "-" * 55)
```

```
    # 시계열 형태의 최종 데이터 생성
    features_ts = df_ts[feat_names].values.astype(np.float32)
    y_values = df_ts["y"].values.astype(np.float32)
    indices = df_ts.index.values

    X_list, y_list, idx_list = [], [], []

    # 시계열 데이터에서 sliding windows 생성
    for end in range(lookback - 1, len(df_ts)):
        start = end - lookback + 1
        X_list.append(features_ts[start : end + 1])  # (lookback, num_channels)
        y_list.append(y_values[end])
        idx_list.append(indices[end])

    X = np.stack(X_list, axis=0)  # (N, lookback, num_channels)
    y = np.array(y_list) # (N,)
    idx = np.array(idx_list)

    # alternative: np.lib.stride_tricks.sliding_window_view 사용
    # X = np.lib.stride_tricks.sliding_window_view(features_ts, window_shape=lookback, axis=0)
    # y = y_values[lookback - 1:]
    # idx = indices[lookback - 1:]

    return X, y, idx, feat_names


# ------------------------------------------------------------
# 멀티채널 Conv1D 모델 (kernel_size=3 커널) 정의 함수
# ------------------------------------------------------------
def build_conv1d_model(lookback: int, num_channels: int):

    model = keras.Sequential([
        layers.Input(shape=(lookback, num_channels)),
        # Conv1D 레이어: 과거 시점의 정보를 활용할 수 있도록 "causal" 패딩 사용
        layers.Conv1D(
            filters=32,
            kernel_size=3,
            padding="causal",
            activation="relu",
        ),
        layers.MaxPooling1D(5),
        layers.Flatten(),
        layers.Dropout(0.2),
        layers.Dense(32, activation="relu", name="hidden"),
        layers.Dropout(0.2),
        layers.Dense(1, activation=None, name='output'),  # 회귀 모델을 위해 선형 활성화 (None)
    ])

    model.compile(optimizer=keras.optimizers.Adam(learning_rate=1e-3), loss="mse", metrics=["mae"])

    return model
```

```
# ------------------------------------------------------------
# MAIN
# ------------------------------------------------------------
if __name__ == "__main__":

    # ------------------------------
    # 0. Raw 데이터 로딩
    # ------------------------------
    df_ts = pd.read_excel('timeseries.xlsx', sheet_name='data', skiprows=13, index_col=0)


    # ------------------------------
    # 1. CNN용 X, y 생성
    # ------------------------------
    X, y, idx, feat_names = make_multichannel_windows(df_ts, lookback=20)

    print("Feature channels:", feat_names)
    print("X.shape:", X.shape)
    print("y.shape:", y.shape, "\n" + "-" * 55)

    # X의 각 채널별 통계량 확인
    # X (samples, lookback, channels) -> (samples * lookback, channels) 2D로 변환
    print("\n--- X, channels description ----------------------------")
    print(pd.DataFrame(X.reshape(-1, X.shape[-1]), columns=feat_names).describe().iloc[1:3])
    print("-" * 55)


    # ------------------------------
    # 2. CNN 모델 학습 및 평가
    # ------------------------------
    tscv = TimeSeriesSplit(n_splits=5, gap=5)

    all_val_losses = []
    all_val_maes = []

    # ------------------------------
    # Time Series Cross Validation (Expanding)
    # ------------------------------
    for fold, (train_idx, val_idx) in enumerate(tscv.split(X), start=1):
        print(f"\n===== Fold {fold}/5 =====")
        print(f"Train index: {train_idx[0]} ~ {train_idx[-1]}")
        print(f"Valid index: {val_idx[0]} ~ {val_idx[-1]}")

        X_train_raw, X_val_raw = X[train_idx], X[val_idx]
        y_train, y_val = y[train_idx], y[val_idx]

        # 데이터 표준화: train 데이터에만 fit하고, train/val 모두 transform
        scaler = StandardScaler()
        ## Reshape, fit, transform, and reshape back for train set
        X_tr_shape = X_train_raw.shape
        X_train = scaler.fit_transform(X_train_raw.reshape(-1, X_tr_shape[-1])).reshape(X_tr_shape)
        ## Reshape, transform, and reshape back for val, test set
        X_val_shape = X_val_raw.shape
        X_val = scaler.transform(X_val_raw.reshape(-1, X_val_raw.shape[-1])).reshape(X_val_shape)
        ## describe
        print("--- X_train (After scaling), channels description -----")
        print(pd.DataFrame(X_train.reshape(-1, X_tr_shape[-1]), columns=feat_names).describe()[:3])


        # 매 fold마다 모델 새로 생성 (가중치 초기화)
        model = build_conv1d_model(lookback=X.shape[1], num_channels=X.shape[2])
```

```
        # 콜백 정의: 조기 종료 및 학습률 스케줄러
        early_stopping_cb = EarlyStopping(patience=10, restore_best_weights=True)
        reduce_lr_cb = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=5, min_lr=0.00001)
        callbacks = [early_stopping_cb, reduce_lr_cb]

        # 학습 (시계열이므로 shuffle=False)
        history = model.fit(
            X_train, y_train,
            validation_data=(X_val, y_val),
            epochs=200,
            callbacks=callbacks,
            shuffle=False,
            verbose=0,
        )

        # 해당 fold 검증 성능 기록
        val_loss, val_mae = model.evaluate(X_val, y_val, verbose=0)
        print(f"Fold {fold} - val_loss: {val_loss:.4f}, val_mae: {val_mae:.4f}")
        all_val_losses.append(val_loss)
        all_val_maes.append(val_mae)


# ------------------------------
# 마지막 Fold의 학습 과정 시각화
# ------------------------------
plt.plot(history.history['loss'], label='loss')
plt.plot(history.history['val_loss'], label='val_loss')
plt.title('Model Loss (Last Fold)')
plt.legend()
plt.show()

# ------------------------------
# 전체 fold 성능 요약
# ------------------------------
print("\n===== Cross-Validation Result (Expanding Window) =====")
print(f"val_loss (per fold): {[f'{v:.4f}' for v in all_val_losses]}")
print(f"val_mae  (per fold): {[f'{v:.4f}' for v in all_val_maes]}")
print(f"mean val_loss: {np.mean(all_val_losses):.4f}")
print(f"mean val_mae : {np.mean(all_val_maes):.4f}")
```

삼성 All That Quant

[팩터 모델 실전 전략 및 이론]

25/10/27 "영업이익 증가율 모멘텀 팩터 - 신규 추천 전략" - High return, Low correlation의 고효율 팩터전략 신규 개발 및 제시
25/8/19 "Long-only 전략 상세 분석법 (롱 팩터 전략 시리즈 #2)" - 효율적 Long-biased 전략을 위한 프로세스와 추천 전략 제시
25/7/21 "Long-only 전략과 비대칭 알파 ((롱 팩터 전략 시리즈 #1)" - 코스피 5000 시대를 대비하는, long-biased 전략 집중 분석
24/11/20 "연구개발 팩터 투자전략" - R&D에 기반한 신규 팩터 전략을 개발하고, 이의 특징과 활용 방법 등을 제시함
24/8/8 "코스닥 투자자를 위한 퀀트 전략 Redux" - 코스닥 시장에서 변화하는 팩터 및 스타일 전략의 특징을 상세 재정리함
24/7/2 "연기금 투자자를 위한 장기 퀀트 전략" - 투자 horizon을 장기로 가져갔을 때의 팩터 투자 전략의 변화를 상세하게 비교 정리함
23/11/30 "퀀트 전략에서의 업종 특성 고려, 2편 - 업종 대비 상대 밸류에이션 팩터의 개선 전략" - 상대 밸류 팩터에서의 업종 특성 고려 전략 제시
23/9/4 "퀀트 전략에서의 업종 특성 고려, 1편 - 저P/B 팩터의 업종 조정 전략" - 한국 업종의 특성을 감안한 수정된 밸류에이션 전략 제시
23/7/26 "Data-driven Quant Model 2 - #연속 국면 #머신러닝 #부스팅 기법 #동적 팩터 모델" - 부스팅 알고리즘 기반의 팩터 로테이션 모델 제시
23/5/17 "퀄리티 팩터는 정말 퀄리티 있는 전략인가" - 퀄리티 팩터가 가지는 고유 특성과 세부 전략 간의 차이점을 비교 정리
23/1/4 "포트 전략 - 롱/숏 시그널로 롱 포트폴리오 구성하기" - 롱/숏 전략을 Long-only 포트로 전환하는 효율적인 알고리즘을 신규 제시
22/11/21 "Data-driven Quant Model - 머신러닝 기법으로 시장의 파도에 올라타기" - 머신러닝 기법에 의한 데이터 기반의 팩터 로테이션 모델 제시
22/8/17 "변동성 구간에서의 주식 스타일 특징 - 낙폭 과대주 팩터 활용법" - 팩터 투자 틀 안에서 낙폭 과대주 팩터의 필요성 입증과 활용법 제시
22/5/25 "경기 국면에 따른 리스크 팩터 배분 전략" - 리스크 팩터 전반에 대한 설명과 함께 팩터와 경기 국면을 결합한 투자 전략을 제시
21/2/24 "이익 모멘텀의 차별화" - 종목 특성을 감안한, 변동성 조정 이익 모멘텀 팩터 전략을 제시
21/1/11 "주식 스타일 Fact Book - 주식 스타일 및 매크로 변수 상관성 팩트 체크" - 주식 스타일 분석 체계의 정리 및 매크로 변수 연결
19/4/15 "팩터 모델의 구조와 해설 - Factor Models for Asset Returns" - 팩터 모델 이론에 대한 상세 설명 자료
18/7/3 "Fama-French 3 팩터 모델 - 후편" - Fama & French의 3팩터 모델 1993년 논문에 대한 전문 번역과 한국 시장 분석 추가 자료
18/5/8 "Fama-French 3 팩터 모델 - 전편" - Fama & French의 3팩터 모델 1992년 논문에 대한 전문 번역과 한국 시장 분석 추가 자료
15/10/20 "가치주 투자와 금리의 연관성" - 금리와 가치/성장 스타일의 인과 분석을, Equity Duration 개념을 활용하여 국내 최초로 제시
14/11/24 "투자 Horizon에 따른 퀀트전략 변화 - Advanced Factor Book" - 1M ~ 6M의 투자 기간별로 팩터 모델 백테스팅 분석
13/4/23 "매크로 기반 Dynamic Quant 모델" - 매크로 국면 판단에 기반한 동적 팩터 투자 모델 정립
12/2/27 "스타일 로테이션의 허와 실" - 스타일 로테이션 전략의 형태로서 모멘텀 방식, 리버전(컨트래리안) 방식, Asness 방식을 각기 비교 분석

[퀀트 전략 일반]

26/1/2 "퀀트 모델링 A to Z - (6) MLP(Multi-Layer Perceptron) AI 투자 모델" - 딥러닝을 활용한 퀀트 투자 모델 개발법을 상세히 설명함
25/11/14 "자산배분 전략의 체계" - 자산배분에 대한 기본적인 개념과 접근법을 체계적으로 정리함
25/9/2 "머신 러닝 애널리스트, 파트 1" - 주식 시장의 기업 펀더멘털 분석에서, 머신러닝 및 AI를 활용하는 접근법 제시
25/7/29 " KR-FinBERT의 활용법" - 센티먼트 분석 AI를 손쉽게 사용하는 방법을 정리함
25/5/20 "Corporate Action Book ver 2.0" - 여러 corporate action 이벤트에 대한 메커니즘 설명과 시장 반응의 최근 패턴을 정리함
25/3/20 "MSCI 지수 업데이트 ver 5.0" - MSCI 지수의 최근 이슈과 방법론 상의 변화, 정기 리뷰 로직의 요약 정리 제공
25/1/23 "MSCI 미국 지수 2월 리뷰 편출입 최종 전망" - 24년말부터 MSCI USA 종목 편출입 전망 리포트를 최초 개시함
25/1/14 "TAX 전략 2. 소득세뿐 아니라 건강보험료도 존재한다" - 개인 투자자 관점에서의 건강보험료 및 광의의 세금에 대한 설명 및 전략 자료
24/11/7 "AI와 퀀트 투자" - AI에 의한 세상의 변화와 알고리즘에 대한 이해, 퀀트 투자와의 접목법을 정리함
24/5/27 "KOSPI 지수 장기 수익률 성과 분해" - KOSPI 장기 수익률을 기업이익 증가, 주식수 변동, 밸류에이션으로 분해하여 부진 원인을 정리
24/5/8 "주당 지표(per share) 기준의 펀더멘털 접근법" - 지수 차원에서의 주당 지표 산출법과 이를 통한 시장 분석, 관련 팩터 전략 등을 정리
24/1/29 "이전 상장 이벤트 및 활용 전략" - 코스피 이전 상장 종목에 대한 주가 변화 결정 요인 상세 분석
23/7/11 "한미 간 자사주 매입의 구조적 차이" - 자사주 매입에 대한 한국과 미국 제도를 비교하고, 시가총액 정의 차이에 따른 문제점을 정리
23/4/17 "NSS(Nelson Siegel Svensson) 모델을 활용한 채권 퀀트 전략, 파트2" - NS 및 NSS 모델을 활용한 실전적인 채권 퀀트 전략 제시
22/11/7 "연방기금 선물 내재 금리의 이해" - 금리선물 등에 기반한 미래 시점별 정책금리 전망치를 확인하는 방법 및 이슈 정리
22/8/1 "주식과 채권의 상관성, 파트2 - 주식과 채권 간 상관 관계의 다이내믹스" - 주식/채권 상관관계의 변화를 경기과 물가 요인으로 설명 및 전망
22/5/17 "퀀트 모델링 A to Z - (5) 포트폴리오 밸류에이션 지표 계산법 (가중 조화평균)" - 가중 조화평균을 활용한 포트폴리오 지표 산출법 정리
20/10/28 "한국이 MSCI 선진시장에 들어가야 하는 이유" - MSCI 선진시장 승격의 당위성과 승격 시의 변화 등에 대해 총정리
19/11/18 "코스피 단기예측 AI 모델 - 랜덤 포레스트 기법을 활용한 머신러닝 기반 모델" - 랜덤 포레스트 기반의 시장 예측 모델 및 소스 코드 제시
17/3/8 "Volatility Drag - 레버리지 투자 수익과 변동성 손실의 관계" - 변동성 손실 개념과 레버리지 상품의 위험성에 대한 상세 설명
16/1/4 "기계학습(Machine Learning)과 투자전략 - 빅 데이터 기법: Naive Bayes Classifier의 활용" - 머신 러닝에 기반한 퀀트 전략 개발
15/7/20 "퀀트 Investment Clock, 한국편" - 성장과 물가 지표를 조합한 4개 국면 기반의 자산배분 모델 제시
14/4/24 "CAPE in Everywhere - Cyclically Adjusted P/E의 다용도 활용 분석" - 한국 증시의 CAPE 지표에 대한 본격 분석
11/5/17 "MSCI 종목편입 효과 실증분석 - 종목 편출입에 따른 최적 투자전략" - MSCI 이벤트 활용 투자 전략의 시작

Appendix. Samsung Q.Pack 메일링 자료

글로벌 주식시장 동향부터, FICC, 경제지표, 국내외 종목 및 ETF 정보까지 수록함. 이메일을 통해 정기적으로 엑셀 자료 전송

Index	Region /Country	Current Index (p)	Index change (%) (a=b+c)								Fwd EPS change (%) (b)				Fwd PER change (%) (c)				EPS Growth (% y-y)			PER (x)			
			1D	2D	1W	MTD	1M	3M	YTD	1Y	1M	3M	YTD	1Y	1M	3M	YTD	1Y	Fwd	21E	22E	Fwd	21E	22E	4yAvg
MSCI	AC World	742.26	0.0	0.0	1.3	4.6	2.5	2.7	14.9	28.0	0.2	4.1	24.6	32.9	2.3	-1.3	-7.9	-3.7	12.2	49.3	7.7	**18.5**	19.7	18.3	16.6
Index	World (DM)	3,151.16	0.0	0.1	1.3	4.8	2.5	3.4	17.1	30.2	0.6	4.7	26.0	34.1	1.9	-1.2	-7.1	-2.9	12.4	48.2	7.9	**19.5**	20.9	19.3	17.3
	N. America	4,627.53	-0.2	0.1	1.6	5.8	3.5	4.1	20.8	32.4	1.0	4.7	28.8	35.6	2.5	-0.6	-6.2	-2.3	12.3	48.0	8.4	**21.5**	23.1	21.4	18.7

Code	Name	연간실적																분기실적						
		매출액(십억)		매출액y-y(%)		영업이익(십억)		영업이익y-y(%)		순이익(십억)		순이익y-y(%)		지배순이익(십억)		지배순이익y-y(%)		매출액(십억)				매출액y-y(%)		
		2021E	2022E	2021E	2022E	2021E	2022E	2021E	2022E	2021E	2022E	2021E	2022E	2021E	2022E	2021E	2022E	1Q21	2Q21	3Q21	4Q21	1Q21	2Q21	3Q21
A005930	삼성전자	276,340	297,907	16.7	7.8	52,686	56,360	46.4	7.0	39,936	43,047	51.2	7.8	38,912	42,428	49.1	9.0	65,389	63,672	73,000	74,295	18.2	20.2	9.0
A000660	SK하이닉스	42,530	49,328	33.3	16.0	12,035	13,823	140.1	14.9	9,128	10,460	91.8	14.6	9,113	10,425	91.7	14.4	8,494	10,322	11,814	11,902	18.0	19.9	45.3
A035420	NAVER	6,784	8,312	27.9	22.5	1,360	1,756	11.9	29.2	15,824	1,825	1,772.6	-88.5	16,582	1,854	1,554.8	-88.8	1,499	1,664	1,727	1,900	-13.5	-12.6	1,000.3
A207940	삼성바이오로직스	1,491	1,690	28.0	13.4	490	531	67.4	8.4	396	435	64.3	9.8	396	435	64.2	9.8	261	412	411	421	25.9	34.0	49.7
A051910	LG화학	44,085	49,876	46.6	13.1	5,849	4,968	225.3	-15.1	4,439	3,476	550.5	-21.7	4,077	3,173	695.2	-22.2	9,650	11,456	11,285	11,733	35.6	74.9	50.3
A035720	카카오	5,987	7,607	44.0	27.1	786	1,147	72.3	46.0	951	1,097	448.6	15.3	902	1,029	479.4	14.1	1,258	1,352	1,644	1,733	44.9	41.9	49.4
A006400	삼성SDI	13,871	16,666	22.8	20.2	1,170	1,577	74.3	34.8	1,194	1,498	89.2	25.5	1,119	1,410	94.6	26.0	2,963	3,334	3,617	3,957	23.6	30.3	17.2
A005380	현대차	116,993	125,659	12.5	7.4	7,136	8,018	198.0	12.4	6,635	7,393	244.7	11.4	6,020	6,795	322.7	12.9	27,391	30,326	28,132	31,054	8.2	38.7	2.0
A068270	셀트리온	2,047	2,419	10.7	18.2	820	963	15.2	17.4	654	762	26.0	16.4	641	746	25.5	16.4	457	432	486	584	22.6	0.7	-11.4
A000270	기아	70,551	75,507	19.2	7.0	5,320	5,854	157.5	10.0	4,733	5,190	218.1	9.7	4,730	5,181	217.9	9.5	16,582	18,339	17,125	18,478	13.8	61.3	4.9
A323410	카카오뱅크					296	521	141.8	75.9	244	399	114.3	64.0	244	399	114.3	64.0	225	254					
A005490	POSCO	72,925	74,829	26.2	2.6	8,884	8,158	269.7	-8.2	6,603	5,877	269.3	-11.0	6,094	5,425	280.4	-11.0	16,069	18,292	20,610	19,280	10.5	33.3	44.5
A012330	현대모비스	41,047	45,462	12.1	10.8	2,228	2,706	21.8	21.4	2,616	3,102	71.3	18.6	2,599	3,081	70.0	18.5	9,816	10,285	9,900	11,004	16.5	36.5	-0.9
A259960	크래프톤	2,128	3,306	27.4	55.4	907	1,531	17.2	68.9	722	1,197	29.8	65.9	722	1,197	29.8	65.9	461	459	460	757	-9.3	21.2	31.4
A028260	삼성물산	32,740	32,926	8.4	0.6	1,407	1,448	64.2	2.9	1,923	1,796	65.7	-6.6	1,730	1,563	67.1	-9.6	7,840	8,548	8,034	8,377	12.6	18.3	2.3
A066570	LG전자	71,956	74,202	13.7	3.1	4,288	4,993	34.2	16.4	2,109	3,675	2.2	74.3	1,821	3,615	-7.5	98.5	18,809	16,117	18,785	17,986	27.7	25.6	11.0
A096770	SK이노베이션	45,612	52,196	33.5	14.4	1,938	2,132	175.5	10.0	901	1,430	142.0	58.8	878	1,401	140.4	59.5	9,240	11,120	12,270	13,120	-17.2	54.5	45.7
A017670	SK텔레콤	19,538	20,303	4.9	3.9	1,526	1,678	13.1	10.0	2,655	2,905	77.0	9.4	2,634	2,834	75.1	7.6	4,780	4,818	4,914	5,036	7.4	4.7	3.9
A105560	KB금융					6,172	6,351	33.7	2.9	4,497	4,688	28.4	4.2	4,463	4,646	29.2	4.1	17,584	12,242	16,473		-8.8	14.2	56.4
A051900	LG생활건강	8,401	9,115	7.1	8.5	1,313	1,463	7.5	11.5	892	1,005	9.7	12.7	874	984	9.6	12.6	2,037	2,021	2,139	2,203	7.4	13.4	3.3
A055550	신한지주					5,993	6,188	21.6	3.3	4,334	4,484	23.9	3.5	4,292	4,414	25.7	2.8	6,895	6,813			-15.3	11.6	
A302440	SK바이오사이언스	976	3,572	332.4	266.2	438	1,578	1,060.7	260.5	355	1,227	980.7	245.2	355	1,227	980.4	245.3	113	145	173	499	397.4	266.8	78.8
A034730	SK	94,268	100,607	15.2	6.7	4,642	5,335	2,922.4	14.9	3,106	3,609	2,964.8	16.2	1,377	1,642	627.2	19.3	22,250	23,406	23,724	25,547	-6.2	24.7	18.9
A091990	셀트리온헬스케어	1,838	2,179	12.9	18.5	331	438	-8.6	32.5	257	336	7.0	30.8	257	336	7.0	30.8	356	433	488	569	-0.2	3.1	5.2
A361610	SK아이이테크놀로	676	1,076	44.1	59.2	189	333	50.7	76.3	148	255	68.1	72.0	149	246	69.3	65.1	139	155	178	227			
A015760	한국전력	59,706	62,797	1.9	5.2	-1,631	-951	-139.9	41.7	-2,165	-1,710	-203.5	21.0	-2,230	-1,755	-212.0	21.3	15,075	13,519	16,039	15,091	-0.1	3.4	2.1
A011200	HMM	12,561	12,447	95.9	-0.9	6,315	5,669	543.9	-10.2	3,837	4,905	2,995.0	27.8	3,866	4,994	3,020.8	29.2	2,428	2,907	3,592	3,489	84.9	111.4	109.0
A003550	LG	8,303	8,612	25.2	3.7	2,797	2,957	64.3	5.7	3,054	2,606	98.3	-14.7	2,958	2,507	101.8	-15.2	2,180	1,924	2,011	2,214	37.5	28.7	2.8
A032830	삼성생명	20,270	20,905	-41.3	3.1	1,182	689	-34.0	-41.7	1,755	1,399	28.1	-20.3	1,653	1,359	30.6	-17.8	10,008	8,195	4,701	4,759	-3.5	7.3	-39.5
A086790	하나금융지주					4,393	4,649	14.5	5.8	3,259	3,416	21.4	4.8	3,200	3,357	21.4	4.9	14,011	6,908	13,195		-32.0	10.8	102.5

Compliance notice

- 보고서는 철저히 계량적 분석에 근거한 의견을 제시합니다. 따라서 당사의 대표 투자의견과 다를 수 있습니다.
- 본 조사분석자료의 애널리스트는 2026년 1월 7일 현재 위 조사분석자료에 언급된 종목의 지분을 보유하고 있지 않습니다.
- 당사는 2026년 1월 7일 현재 위 조사분석자료에 언급된 종목의 지분을 1% 이상 보유하고 있지 않습니다.
- 본 조사분석자료에는 외부의 부당한 압력이나 간섭 없이 애널리스트의 의견이 정확하게 반영되었음을 확인합니다.
- 본 조사분석자료는 당사의 저작물로서 모든 저작권은 당사에게 있습니다.
- 본 조사분석자료는 당사의 동의 없이 어떠한 경우에도 어떠한 형태로든 복제, 배포, 전송, 변형, 대여할 수 없습니다.
- 본 조사분석자료에 수록된 내용은 당사 리서치센터가 신뢰할 만한 자료 및 정보로부터 얻어진 것이나, 당사는 그 정확성이나 완전성을 보장할 수 없습니다. 따라서 어떠한 경우에도 본 자료는 고객의 주식투자의 결과에 대한 법적 책임소재에 대한 증빙자료로 사용될 수 없습니다.
- 본 조사분석자료는 기관투자가 등 제3자에게 사전 제공된 사실이 없습니다.

삼성증권

삼성증권주식회사
서울특별시 서초구 서초대로74길 11(삼성전자빌딩)
Tel: 02 2020 8000 / www.samsungpop.com

삼성증권 Family Center: 1588 2323

고객 불편사항 접수: 080 911 0900

Member of
Dow Jones
Sustainability Indices
Powered by the S&P Global CSA