利用CNN网络对期货数据预测一段时间的未来20天涨跌

参考项目：

<https://github.com/RichardS0268/CNN-for-Trading/tree/main?tab=readme-ov-file>

租用服务器跑深度学习的网站（恒源云）

[恒源云\_GPUSHARE-恒源智享云](https://gpushare.com/center/hire)

运行主体流程：（有点杂乱尚未整理）



本周工作：

①数据集换成期货数据

②按时序划分训练集验证集

③搭建网络

④在2021-2022期货数据集上测试，准确度达51%，有待提高

目录

[1原理 1](#_Toc12956)

[2环境配置 2](#_Toc19122)

[3 文件架构 3](#_Toc23420)

[4数据集预处理 3](#_Toc30638)

[5 划分训练验证集 8](#_Toc27224)

[6 构建网络、训练、测试、推理 8](#_Toc25545)

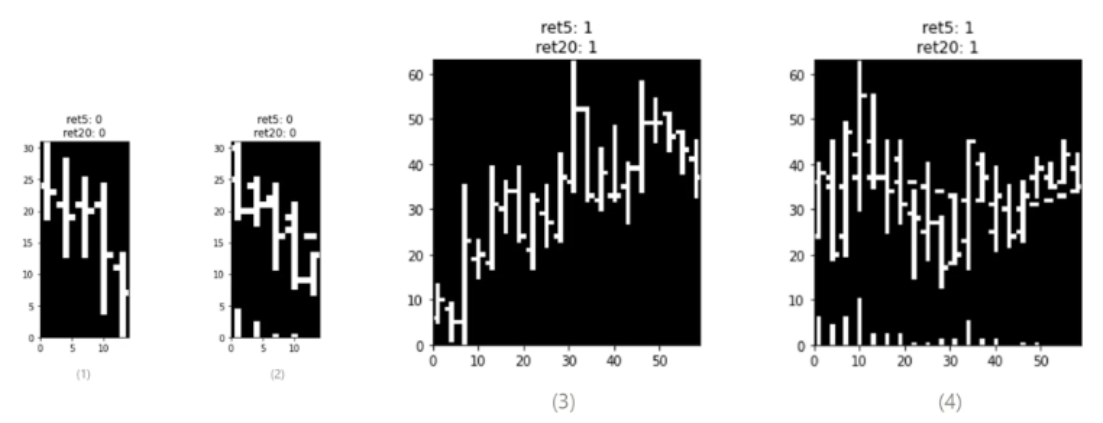
[7 总结与反思 11](#_Toc19864)

[8 未来展望 11](#_Toc28281)

1原理

训练前n年的期货数据，按每20天绘制一张ohlc图（加上MA+volume），同时打上标签，这里一张图的标签是这张图的未来20天的收益率可能上涨还是下跌的趋势（0/1：up/down），将图传入CNN网络进行训练

这样一来，训练之后，当我们给出最近这个月20天的期货数据，就能得到未来20天可能上涨还是下跌的趋势



|  |  |
| --- | --- |
| 20200102-20200206，未来20天可能涨 | 20200107-20200211 |

CNN的输入通常是一幅图像，在我们的设置中，图像是过去市场信息(开盘价、高点、低点和收盘价以及交易量,MA指标)的图，以黑白像素值的矩阵表示。CNN旨在从图像中产生预测，而不需要研究人员手动设计预测特征。并且，CNN将特征提取过程自动化。在给定的“层”中，CNN在空间上平滑图像内容，以减少噪声，并强调与未来回报相关的形状配置。这种平滑操作通过将多层叠加在一起递归地应用，这使CNN能够灵活地捕获潜在的复杂预测模式(并为其赢得了“深度学习”的同义词)。最终的预测集由原始数字像素值的平滑非线性变换组成。在高层次上，这类似于回归分析中使用的更传统的基于核的数据过滤器。然而，CNN并不是简单地修复一组平滑滤波器:相反，它估计出最能检测图像形状和其他属性的滤波器，这些滤波器最能预测目标变量。

简而言之，我们使用CNN是因为它能够从原始输入数据中自动提取预测信号，这使得它非常适合引出金融市场背后的模式，但对于人类来说可能过于复杂而无法假设。

为什么将市场数据编码为图像而不是更标准的时间序列数字格式是有益的?（1）领先的CNN架构是为图像分析定制的。因此，为了享受CNN自动信号生成的好处，用CNN自然吸收的格式表示价格数据是有用的，即作为图像。

1. 将时间序列表示为图像允许模型专注于难以梳理的数据的关系属性

2环境配置

跑深度学习需要比较好的显卡，这里选择租用了恒源云服务器



框架：Pytorch2.0.0+cuda11.8+Python3.8

3 文件架构

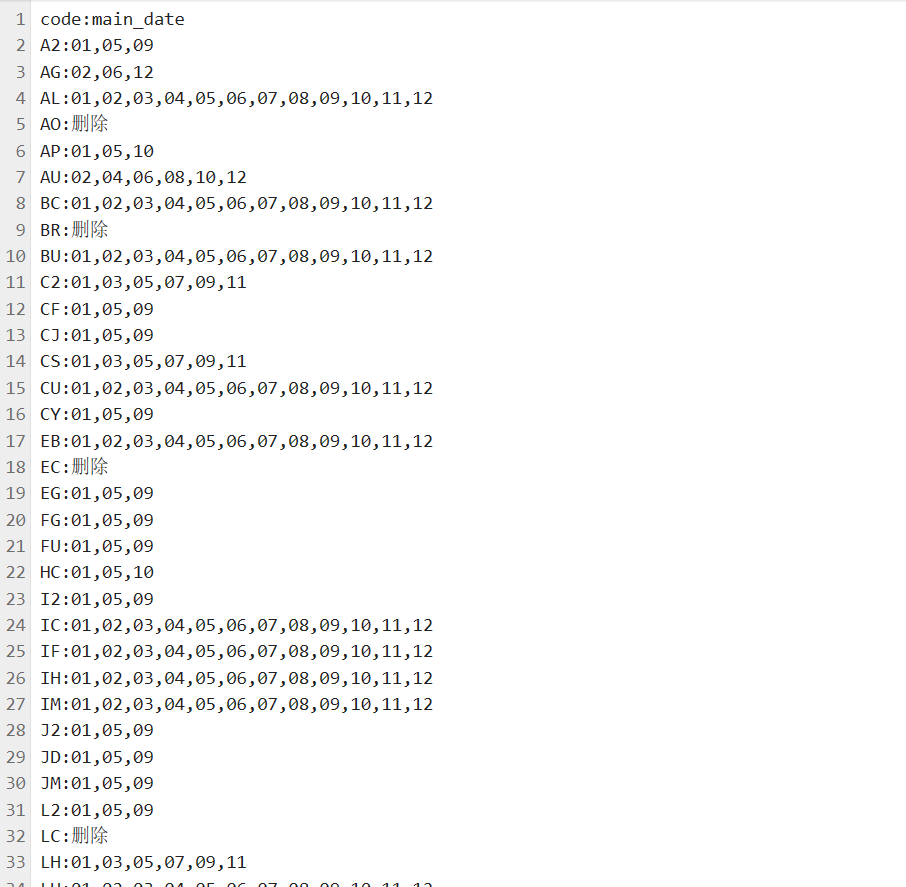
|  |
| --- |
| - dataset ：原始项目的数据集  - data ：原始项目的数据集  - logs ：存储训练日志的位置  - configs：主要的一些配置文件  - models ：生成的模型存储的位置  - model. py ：网络模型文件  - output\_file. csv  - output\_file1. csv ：我们的期货 csv 文件，用于做数据集  - pipeline\_infer. ipynb  - pipeline. ipynb  - pipeline2. ipynb  - pipeline3. ipynb ：运行的主体流程  - test. py：测试文件  - inference. py ：推理文件  - train. py ：训练文件  - dataset. py ：处理数据文件  - main. py ：主函数  - 主力. txt：各种品种的主力月  - data\_clear. ipynb ：处理数据的 ipynb 文件  - splitdata. py：自己写的按每种期货品种指定日期划分的训练测试集  - utils. py  - build. sh  - dataset\_inferance. py  - model\_reference. py  - pipeline1. ipynb  - imgs  - \_\_pycache\_\_  - \_\_init\_\_. py  - README. md  - DayDatas 单品种 2010-2023 ：原始待处理我们服务器上的数据集  - run. ps1  - DayDatas 单品种  - 2010-2023. zip |

4数据集预处理

4.1 预处理

初始数据集是我们服务器上的数据（root/CNN-for-Trading/DayDatas单品种2010-2023.zip），但数据集比较杂且多，有多个csv文件，每个csv文件代表一个期货品种的某一个月的数据

这里我先根据同花顺期货查找了所有期货品种的主力月，生成了一个txt文件，见root/CNN-for-Trading/主力.txt（见下图，用于作为后面筛选的文件）



随后调用root/CNN-for-Trading/data\_clear.ipynb文件，对csv进行合并并筛选最后得到一个总的csv文件（见下图，root/CNN-for-Trading/output\_file1.csv），方便之后训练传入网络中

正式开始流程：root/CNN-for-Trading/pipeline3.ipynb

-code:期货名称

-date：时间

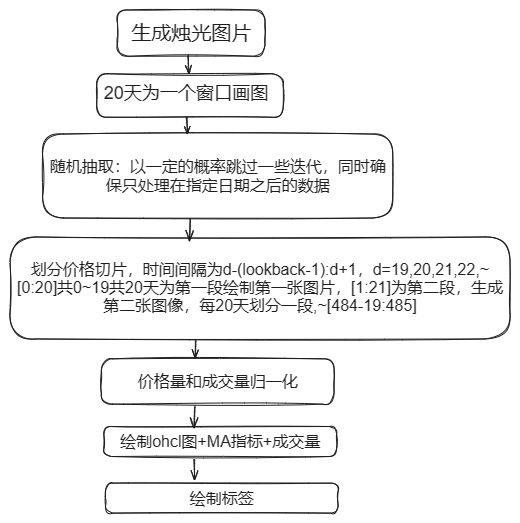
其中open/high/low/close/volume是主要用于后面绘制图像的特征



我们的数据集中一共有251971行数据，1520种期货品种

4.2 时序数据转换为图片

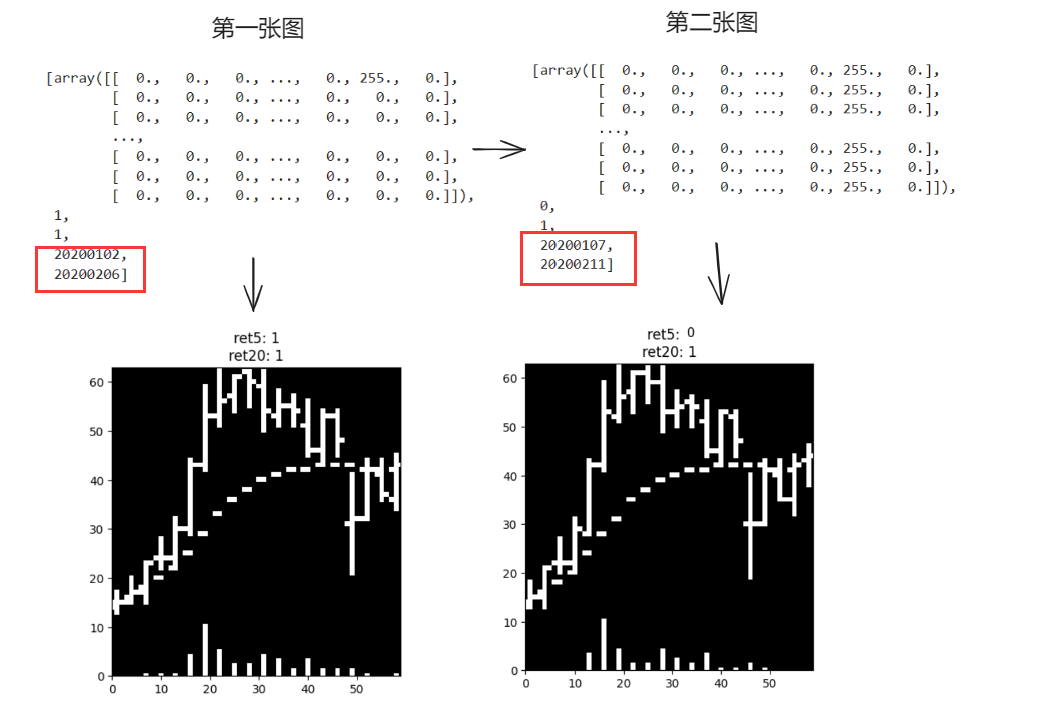
这步的步骤主要见root/CNN-for-Trading/dataset.py



4.3 窗口切片

这里设置的窗口（lookback=20）即训练传入的一张图片包含20天的交易数据，例如从第1天开始，[0:20]绘制一张图片，[1:21]数据绘制下一张图片。

其中，还进行了部分改进，上面绘制的图片有大量重叠的数据，会造成冗余，这里采用了生成随机数，随机跳过部分数据，不仅避免了图片之间信息重复大，而且减少了数据集的训练量。

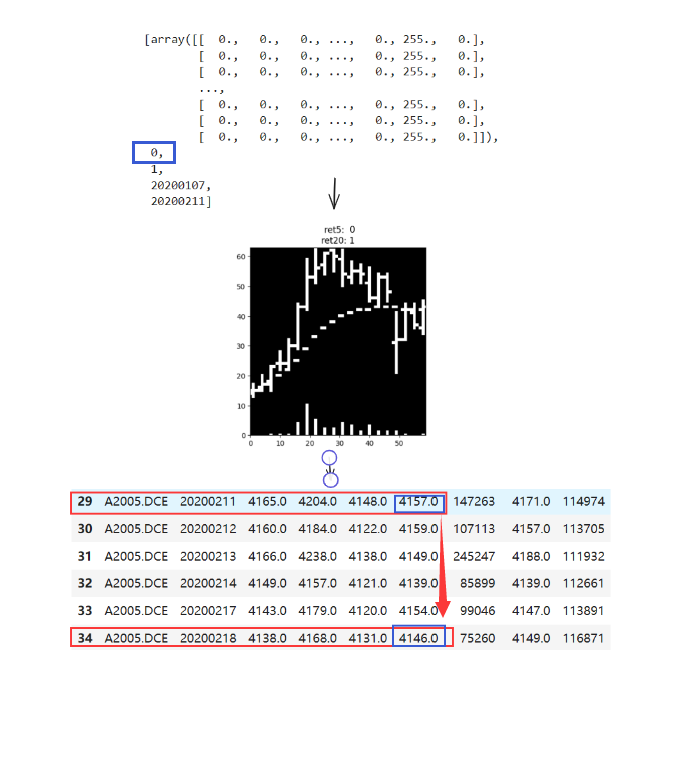


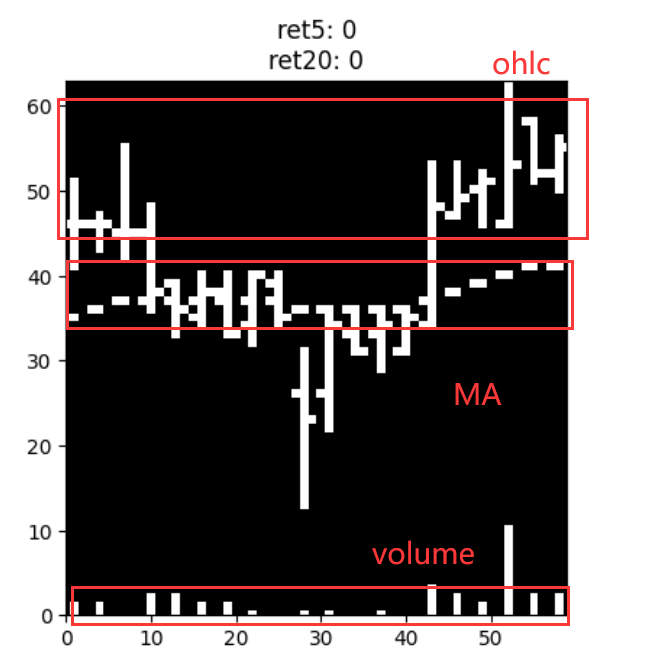
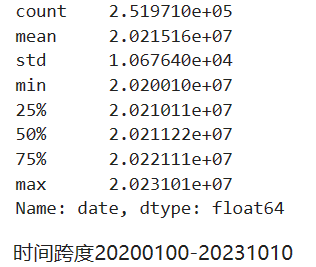
4.4 绘制图片标签

训练集中，一张图片对应着一个标签（ret5/ret20）：0或1，而每张图片都有一个valid\_date，就是一张图片的最后一天的日期，这张图片的标签就是当前图片valid\_date的后5天的收盘价相对于当前图片里最后一天的收盘价的涨跌情况，若为负数，则为0，若为整数则为1

这种标签有很好的意思，当出现这种图片时，可以分析当前最后一天的未来第5天的涨跌情况，从而判断，是买入还是卖出，见下图

这张图的最后一天是20200211，标签显示这天后未来第五天收盘价会比这天低





这里我们的数据集的时间跨度是从2020.01.01-2023.10.10

5 划分训练验证集

这步的步骤详见root/CNN-for-Trading/splitdata.py

5.1采样率

由于计算资源的限制，我们在准备数据集时定义了采样率（sample\_ratio）。

范围为0-1，越靠近1，采样越多，代表用于训练的训练验证集数量越多

在图像生成过程中实际上有两层循环。

外部迭代股票，内部迭代交易日期。

在内部循环中应用采样率: 每次迭代都有跳过的可能性 (采样率)。通过这种方式，我们减少了数据集的大小。内循环的随机抽样迭代是合理的，因为对于相同的股票，相邻天的图像中存在大量重复信息。

这里设置的采样率是0.6，后续可以继续加大，扩充数据集。

5.2如何划分训练验证集

通过循环计数得知，当设定划分日期为20210601时，训练验证集能近似等于7:3

这里的划分不是按照数据集索引的划分，而是设置了一个双重循环，对每一只期货品种来看，有多天数据即每一只期货可生产k张ohlc图片。

我们抽取每只期货的以20210601前的图片做训练集（每只期货20200101-20210601），每只期货的以20210601后的图片做验证集（每只期货20210601-20221231）

目前，总训练验证集一共42819张图片

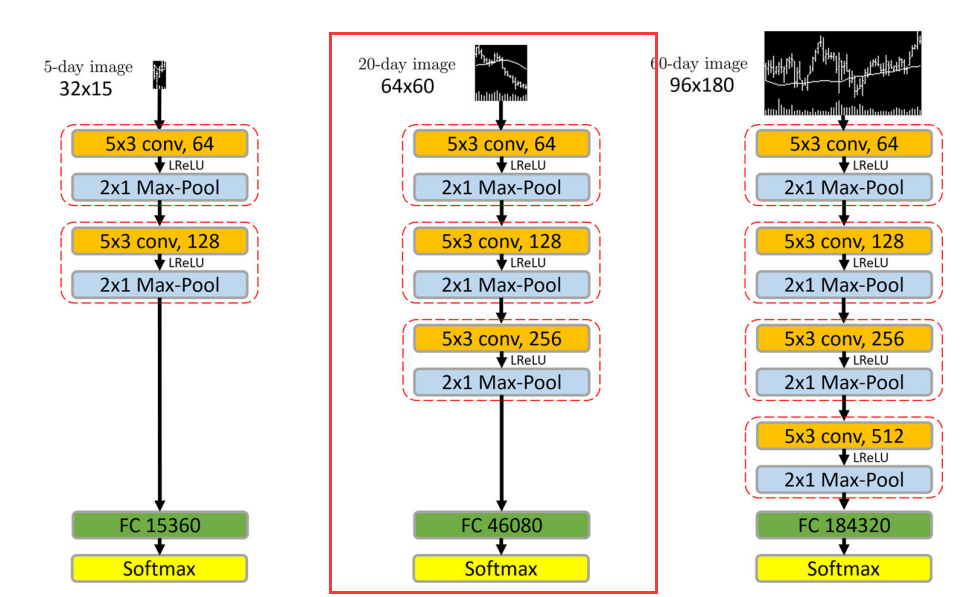
6 构建网络、训练、测试、推理

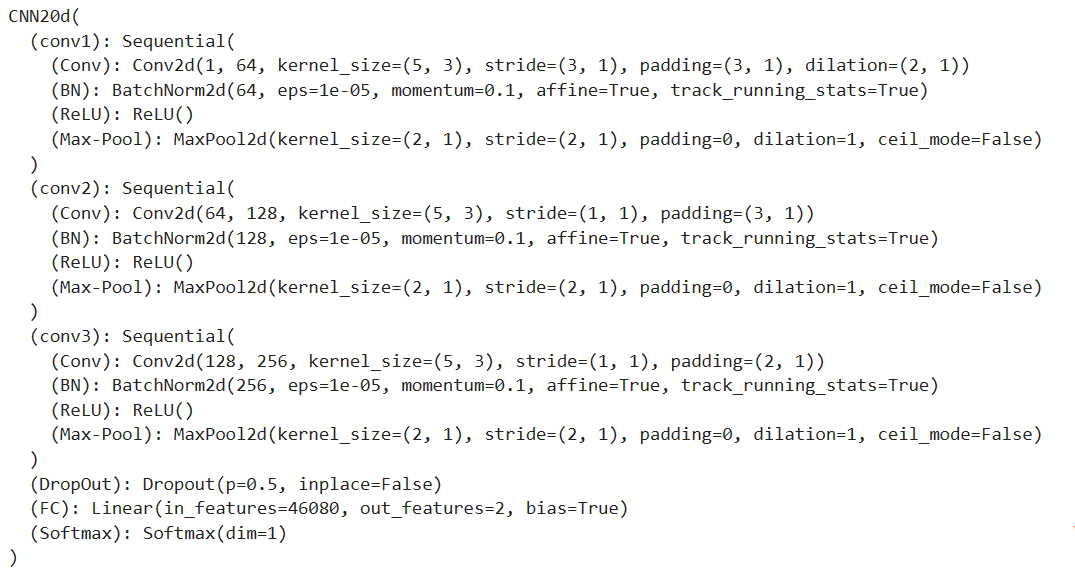
6.1 构建网络（见model.py）

该图显示了5天(左)、20天(中)和60天(右)CNN模型的示意图。5/20/60天CNN模型使用2/3/4个CNN构建块构建，如图所示。符号H x W表示CNN构建块的输出大小，其中H是高度，W是宽度。

conv或MaxPool后面的数字是深度(通道数，例如64、128、256和512)。最后一个CNN块的输出被平面化为一个向量，并馈送到一个全连接(FC)层，其中报告的输入大小(在FC之后)被计算为最后一个CNN构建块的乘积H xW x D。最后的Softmax层产生“上”和“下”的概率。

这里我们选用20天生成一张图的网络搭建





|  |
| --- |
| class CNN20d(nn.Module):  # Input: [N, (1), 64, 60]; Output: [N, 2]  # Three Convolution Blocks,三个卷积块  1.定义初始化权重（权重是干啥的？）  def init\_weights(self, m):  if isinstance(m, nn.Linear) or isinstance(m, nn.Conv2d):  torch.nn.init.xavier\_uniform(m.weight)  m.bias.data.fill\_(0.01)  def \_\_init\_\_(self):  super(CNN20d, self).\_\_init\_\_()  2.第一个卷积块   * 输入通道：1，输出通道：64。 * 卷积核：(5,3)，padding：(3,1)，步长：(3,1)，空洞率：(2,1)。 * 输出尺寸：[N, 64, 10, 60]。 * 卷积-》批归一化-》ReLU-》最大池化（核(2×1)）   self.conv1 = nn.Sequential(OrderedDict([  ('Conv', nn.Conv2d(1, 64, (5, 3), padding=(3, 1), stride=(3, 1), dilation=(2, 1))), # output size: [N, 64, 21, 60]  ('BN', nn.BatchNorm2d(64, affine=True)),  ('ReLU', nn.ReLU()),  ('Max-Pool', nn.MaxPool2d((2,1))) # output size: [N, 64, 10, 60]  ]))  self.conv1 = self.conv1.apply(self.init\_weights)  3.第二个卷积块   * 输入通道：64，输出通道：128。 * 卷积核：(5,3)，padding：(3,1)，步长：(1,1)。 * 输出尺寸：[N, 128, 6, 60]。   self.conv2 = nn.Sequential(OrderedDict([  ('Conv', nn.Conv2d(64, 128, (5, 3), padding=(3, 1), stride=(1, 1), dilation=(1, 1))), # output size: [N, 128, 12, 60]  ('BN', nn.BatchNorm2d(128, affine=True)),  ('ReLU', nn.ReLU()),  ('Max-Pool', nn.MaxPool2d((2,1))) # output size: [N, 128, 6, 60]  ]))  self.conv2 = self.conv2.apply(self.init\_weights)  4.第三个卷积块   * 输入通道：128，输出通道：256。 * 卷积核：(5,3)，padding：(2,1)。 * 输出尺寸：[N, 256, 3, 60]。   self.conv3 = nn.Sequential(OrderedDict([  ('Conv', nn.Conv2d(128, 256, (5, 3), padding=(2, 1), stride=(1, 1), dilation=(1, 1))), # output size: [N, 256, 6, 60]  ('BN', nn.BatchNorm2d(256, affine=True)),  ('ReLU', nn.ReLU()),  ('Max-Pool', nn.MaxPool2d((2,1))) # output size: [N, 256, 3, 60]  ]))  self.conv3 = self.conv3.apply(self.init\_weights)  5.全连接层与输出   * Dropout 层（概率 0.5）。 * 全连接层将 256 \* 3 \* 60=46080 维特征映射到 2 维输出。 * 使用 Softmax 输出概率。   self.DropOut = nn.Dropout(p=0.5)  #输出层  self.FC = nn.Linear(46080, 2)  self.init\_weights(self.FC)  self.Softmax = nn.Softmax(dim=1)  #前向传导，用于分类任务，因为你使用了softmax  def forward(self, x): # input: [N, 64, 60]  x = x.unsqueeze(1).to(torch.float32) # 添加通道维度output size: [N, 1, 64, 60]  x = self.conv1(x) # output size: [N, 64, 10, 60]  x = self.conv2(x) # output size: [N, 128, 6, 60]  x = self.conv3(x) # output size: [N, 256, 3, 60]  x = self.DropOut(x.view(x.shape[0], -1))#展平后Dropout  x = self.FC(x) # output size: [N, 2]#输出[N,2]  x = self.Softmax(x)#概率归一化，输出概率，如果是二分类[0.982, 0.018]，各部分和为1  return x |

6.2 训练(见train.py)

训练过程实际上是把一张张图片传入网络，将输出的和真实的target标签计算loss并不断反向传导，使loss不断减少，即使网络的输出不断逼近真实标签target。

训练还有一个很重要的部分就是调参，见config文件夹中的配置文件

其中，有这样一些超参数使有待调整的：

·NEPOCH: 100

·LEARNING\_RATE: 0.000001

·WEIGHT\_DECAY: 0.001

·EARLY\_STOP\_EPOCH: 6

- 每层权重的初始化器 Adam 优化器

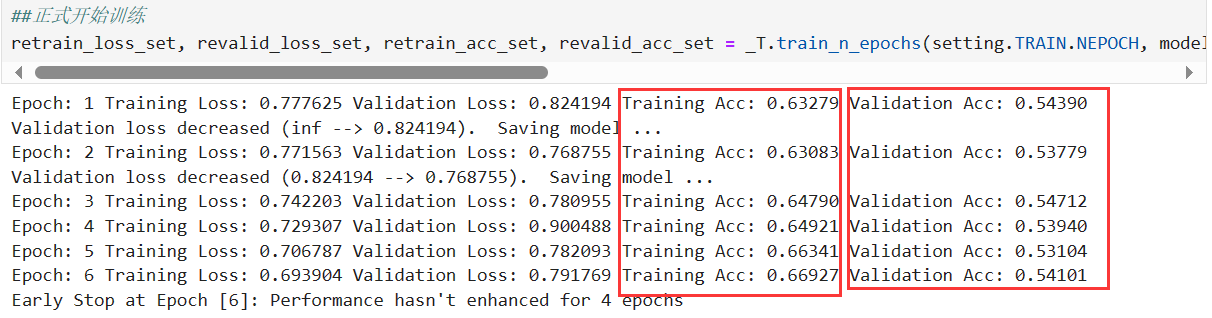
- 权重衰减= 0.01，

- 学习率= 1e-5 卷积

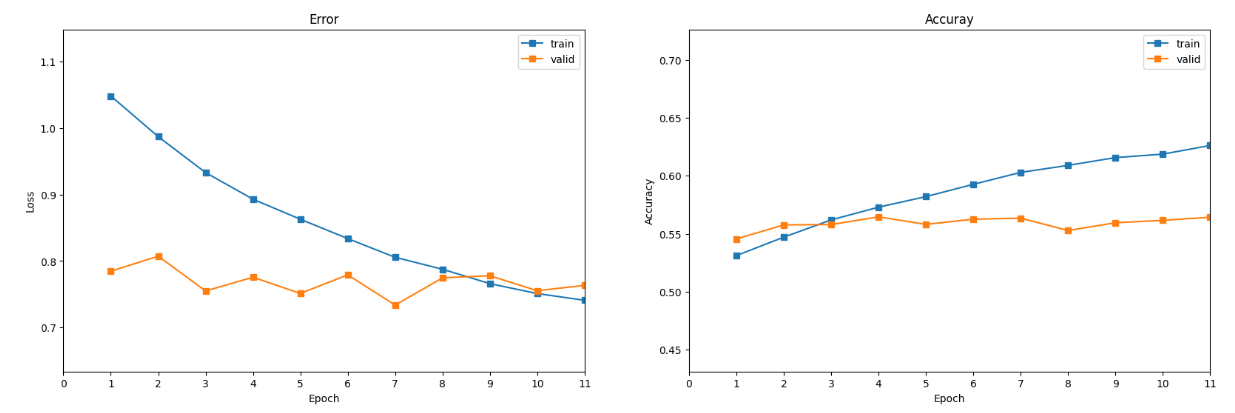
- 非线性激活函数之间的批归一化 FC 层

- 50%掉线提前停止，当连续 16 次验证样本损失未减少时停止训练

最终训练结果生成模型，得到训练和验证的准确度，还有待优化

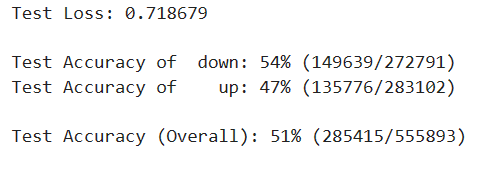


训练损失和准确率：



6.3 测试

训练完成后，生成模型，我们需要验证生成的模型在未使用过的数据集上的准确性，这里采用的数据集是20221231-20231231的数据



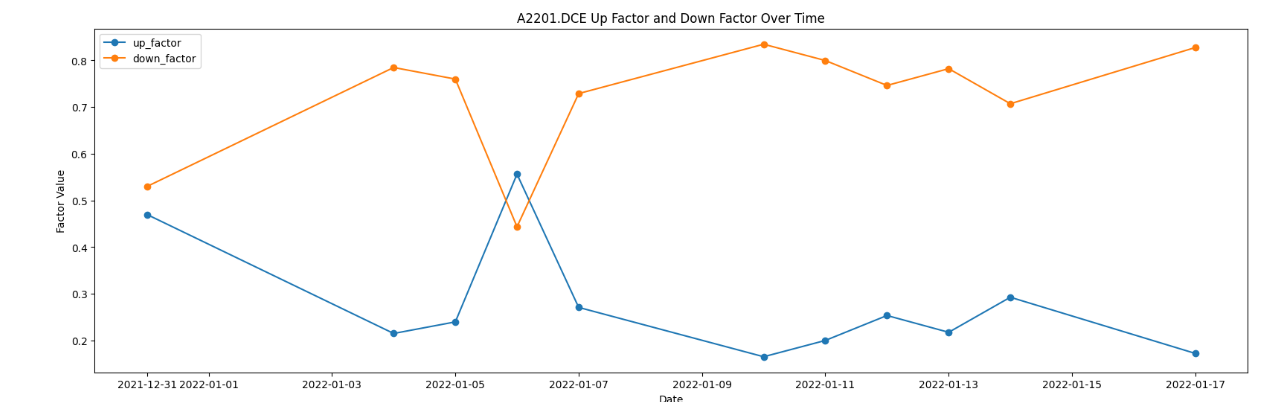
6.4 推理

拿A2201.DCE的20211231-20221231的数据来看，

该图的纵坐标代表的是当前日期下未来的第20天收盘价相对于当前日期的收盘价涨跌概率

·蓝色线是当前日期下未来20天可能上涨的概率，黄色线是当前日期下未来20天可能上涨的概率

·可见**A2201.DCE的2022年1月的后半个多月多呈现下跌趋势**



7 总结与反思

尽管目前结果还不是很好，改进方法：

①数据量不够我们只用了2年的数据训练（实际也就只有251971条数据）

√已做，训练集数据增加到了10万张图片

②采样率过低，这里我的采样率是0.1，这导致用于训练的图片不多，后期可以继续加大

√已做，采样率是0.1，训练集数据增加到了10万张图片

③需要继续调参优化

√已做，调整了LEARNING\_RATE、WEIGHT\_DECAY、EARLY\_STOP\_EPOCH三项参数

④整个项目过程可能还需在疏漏的bug和不符合期货市场的问题，有待后期找到

问题并且优化

->待完善

⑤生成图像只含有MA指标，如果再加些其他指标可能训练效果更好

->待完善

⑥过采样处理不平衡问题

->待完善

8 未来展望

虽然准确度还未能提上去，但相信当准确率提高了后，我们给出当前20天的k线图，网络能返回未来的第5天相对于当天是涨还是跌，这是一件很有意义的事情。

例如，当网络输出的是1，代表未来第5天相对于今天是涨的，那么我们今天可以买入，持仓5天或者更久再卖出，则很大可能会赚钱。

另外，相对于之前用时序数据对未来的数据进行预测，我们这次用到图片信息预测，这也是很有意义的，仅仅只看时序信息会错过，他们之间的关联性，常见的炒股人一般也是分析k线图，而不是只看数据。