Paper Reproduction: (Re)Imag(in)ing price trend

Aoran LI, Tianying ZHOU, Langting WENG, Yijia MA

START

Topic

1. Data Set & Model Construction

2. Model Training

3. Model Performance

4. Conclusion & Future work

Abstract

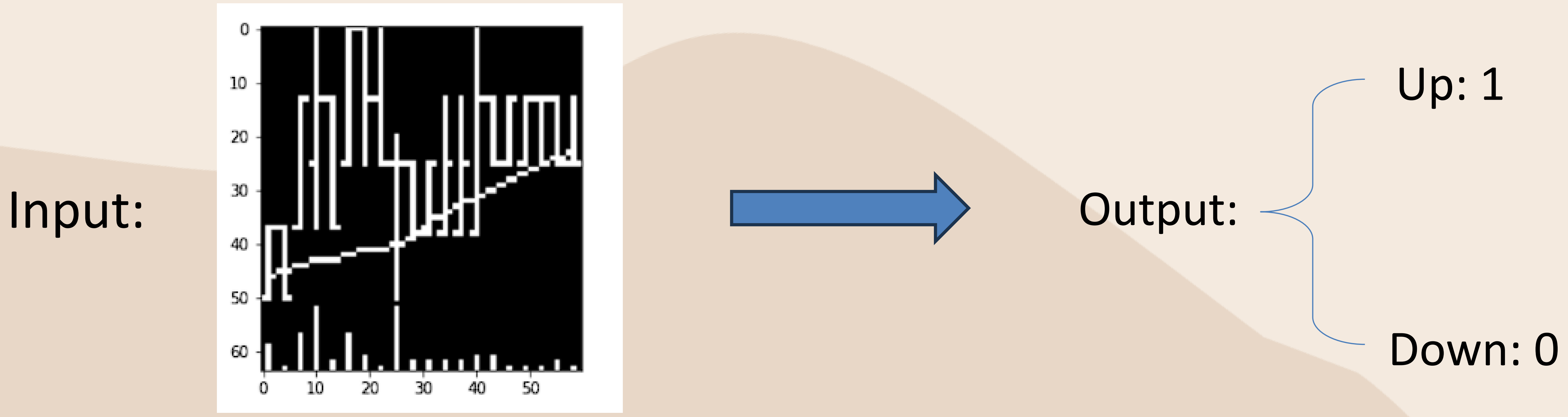
In this experiment, I have re-implemented the main idea of Jiang, Jingwen and Kelly, Bryan T. and Xiu, Dacheng in their paper (Re-)Imag(in)ing Price Trends.

For the data-set, I only use part of the original paper's data, which include the 20 days OHLC data image and the corresponding labels (ret20). The images are two dimensions array, which represent the candle-list figures of previous 20 days when displayed in grey-figures.

For the models, I rigorously follow the architecture and hyper-parameters of the original paper. I have implied the 20-day CNN model proposed by the authors. Apart from the baseline 2 classification label, we implement the same model on further 2 different classification labels, which are ret20 up/down/na and ret5 up/down. So there are total 3 kinds of models in our whole project, i.e. 20D3C, 20D2C, 5D2C.

For the training and inference workflow, we mainly follow the setting of original paper. However, due to limitation of computation resource, we only tried 3-fold model combination.

In the end, I apply our models on the test data-set and discussed the further improvement of our study. The project code is available **My Github Repo**. Please feel free to give a star if you find it useful.



1

Data Set & Model Construction

Data Set

In the original paper, authors construct datasets consisting 3 scale of horizons(5-day, 20-day, 60-day). In this project, I mainly focused on the 20-day image data

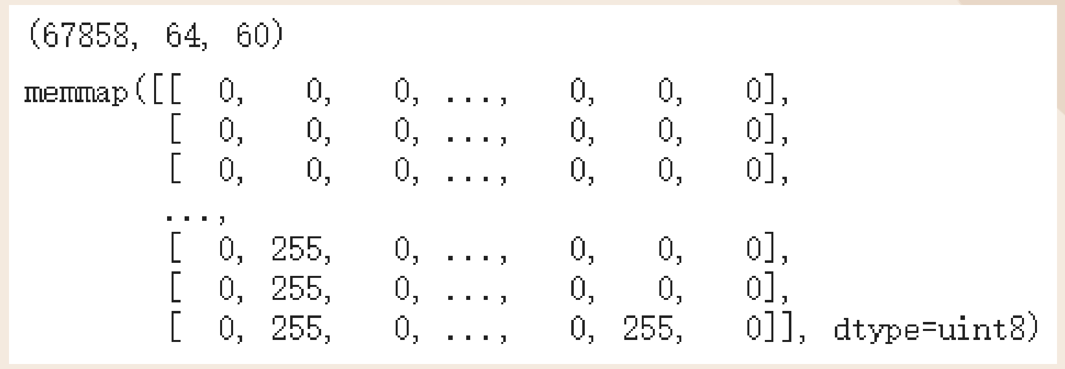
Input Data:

The input image shape is **(64, 60)** with only one channel.

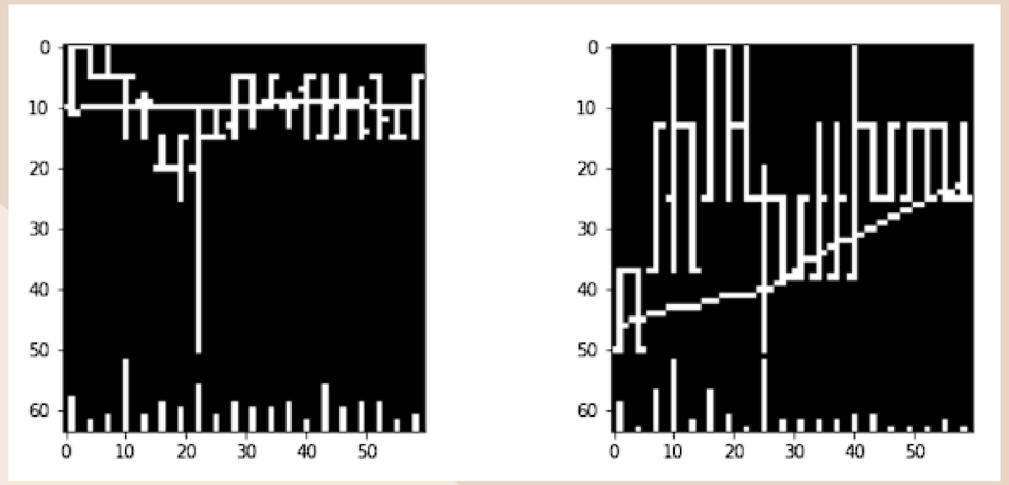
- **Training set:** 1993-2001, 885004, 3.16G
- **Testing set:** 2002-2019, 1311990, 4.69G

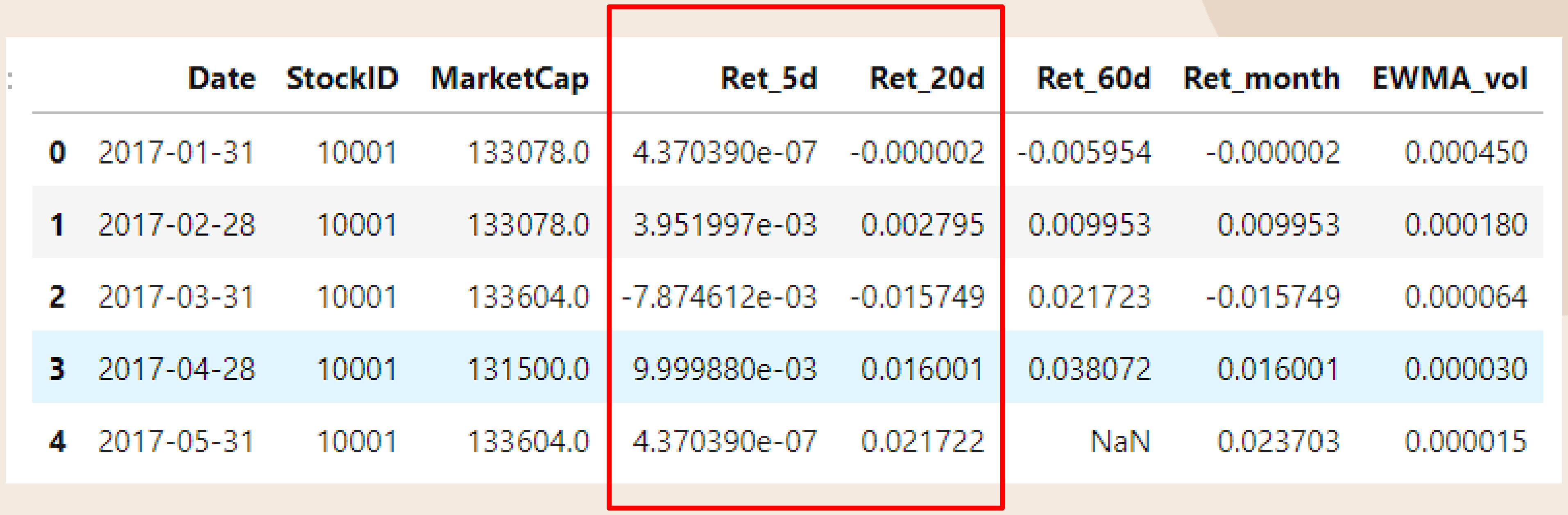
Label:

The corresponding image label is **binary classification** number that represents the positive or negative return in future periods. We have trained **3** models on different labels used **Ret_5d** and **Ret_20d**.



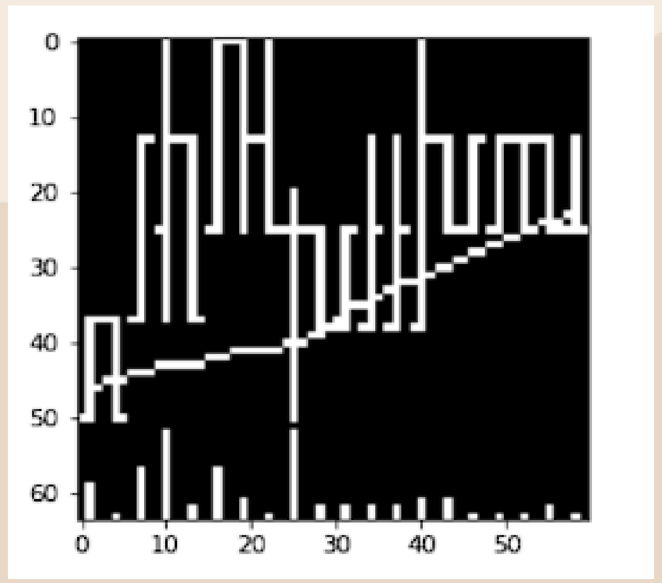
```
(67858, 64, 60)
memmap([[  0,    0,    0, ...,    0,    0,    0],
        [  0,    0,    0, ...,    0,    0,    0],
        [  0,    0,    0, ...,    0,    0,    0],
        ...,
        [  0,  255,    0, ...,    0,    0,    0],
        [  0,  255,    0, ...,    0,    0,    0],
        [  0,  255,    0, ...,    0,  255,    0]], dtype=uint8)
```





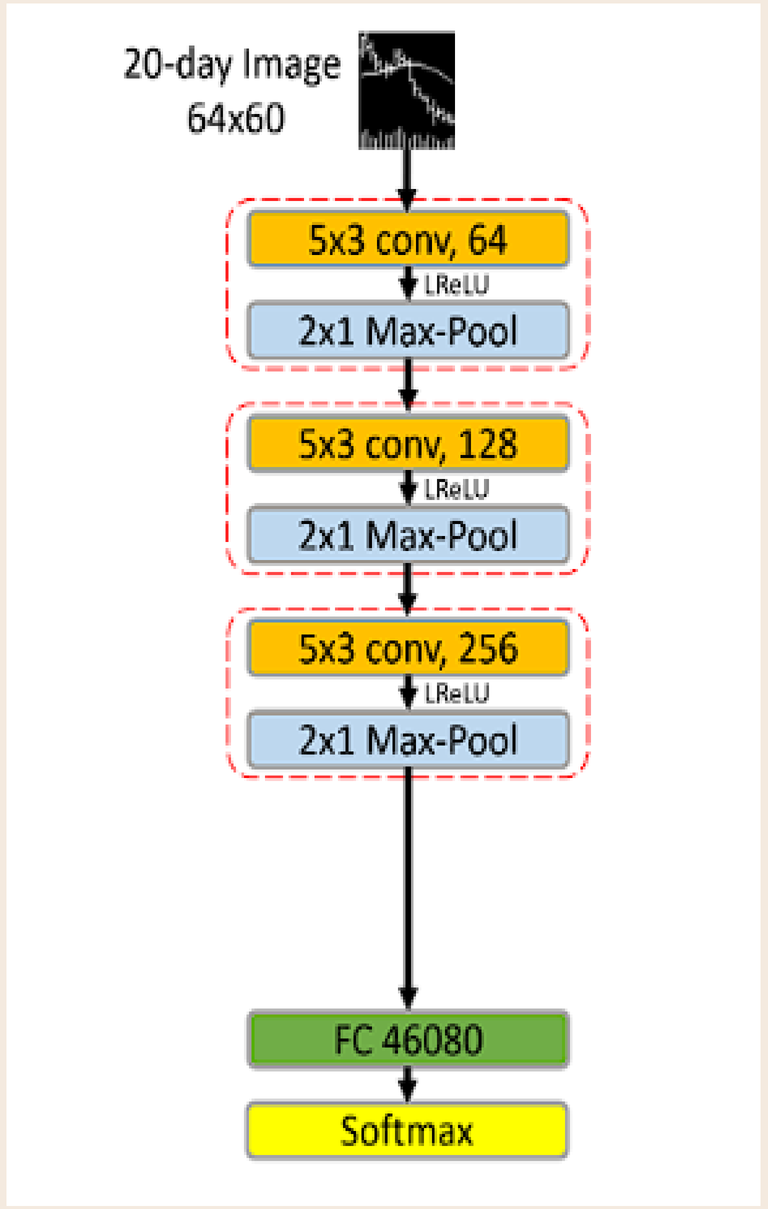
	Date	StockID	MarketCap	Ret_5d	Ret_20d	Ret_60d	Ret_month	EWMA_vol
0	2017-01-31	10001	133078.0	4.370390e-07	-0.000002	-0.005954	-0.000002	0.000450
1	2017-02-28	10001	133078.0	3.951997e-03	0.002795	0.009953	0.009953	0.000180
2	2017-03-31	10001	133604.0	-7.874612e-03	-0.015749	0.021723	-0.015749	0.000064
3	2017-04-28	10001	131500.0	9.999880e-03	0.016001	0.038072	0.016001	0.000030
4	2017-05-31	10001	133604.0	4.370390e-07	0.021722	NaN	0.023703	0.000015

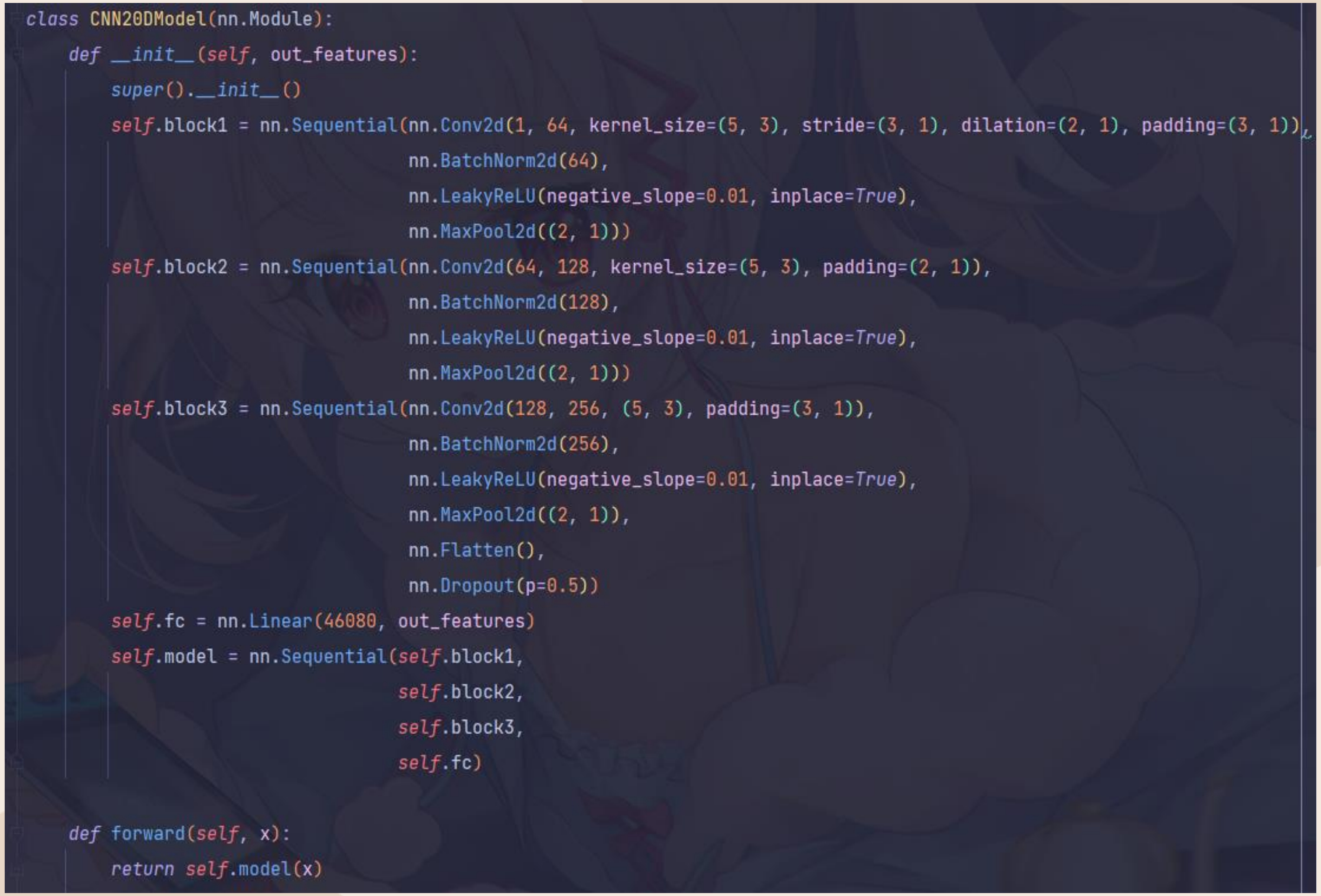
Input:



Output: Up: 1 / Down: 0

Model Construction





```
class CNN20DModel(nn.Module):
    def __init__(self, out_features):
        super().__init__()
        self.block1 = nn.Sequential(nn.Conv2d(1, 64, kernel_size=(5, 3), stride=(3, 1), dilation=(2, 1), padding=(3, 1)),
                                    nn.BatchNorm2d(64),
                                    nn.LeakyReLU(negative_slope=0.01, inplace=True),
                                    nn.MaxPool2d((2, 1)))
        self.block2 = nn.Sequential(nn.Conv2d(64, 128, kernel_size=(5, 3), padding=(2, 1)),
                                    nn.BatchNorm2d(128),
                                    nn.LeakyReLU(negative_slope=0.01, inplace=True),
                                    nn.MaxPool2d((2, 1)))
        self.block3 = nn.Sequential(nn.Conv2d(128, 256, (5, 3), padding=(3, 1)),
                                    nn.BatchNorm2d(256),
                                    nn.LeakyReLU(negative_slope=0.01, inplace=True),
                                    nn.MaxPool2d((2, 1)),
                                    nn.Flatten(),
                                    nn.Dropout(p=0.5))
        self.fc = nn.Linear(46080, out_features)
        self.model = nn.Sequential(self.block1,
                                   self.block2,
                                   self.block3,
                                   self.fc)

    def forward(self, x):
        return self.model(x)
```

The model architectures are the same as in the original paper, which is shown in right figure.

Apart from the basic layer architecture, we also customize some parameters in CNN layer to have same input shape in Linear layer, such as padding parameter in nn.Conv2d.

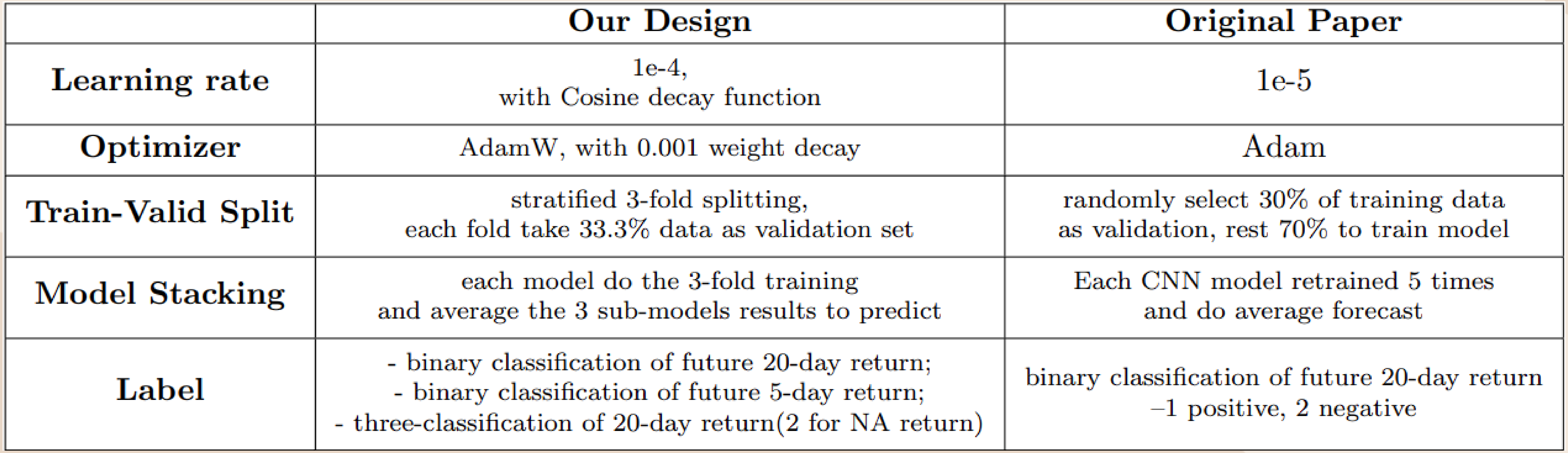
2

Model Training

Training Workflow

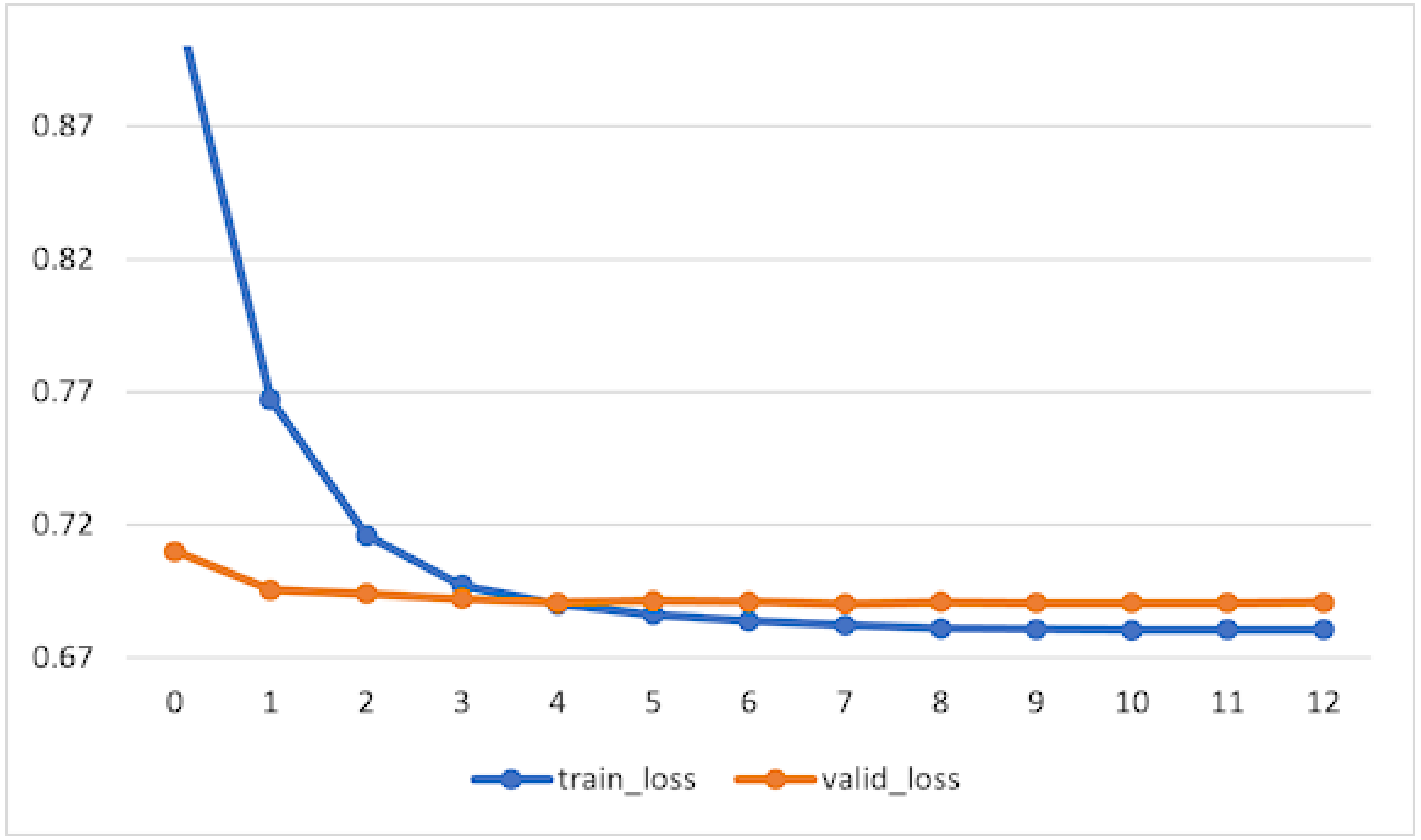
In the model training phase, we adhere to the procedure outlined in the original paper, such as the **xavier initiation**, **batch-normal & drop out** design and **early-stopping mechanism**.

However, due to constraints in computing resources and the limited GPU usage time on Kaggle, we have simplified some of the initial training design. Furthermore, in order to enhance training stability and bolster model robustness, we have implemented certain process improvements. The disparities between our approach and that described in the original article are summarized in Table.

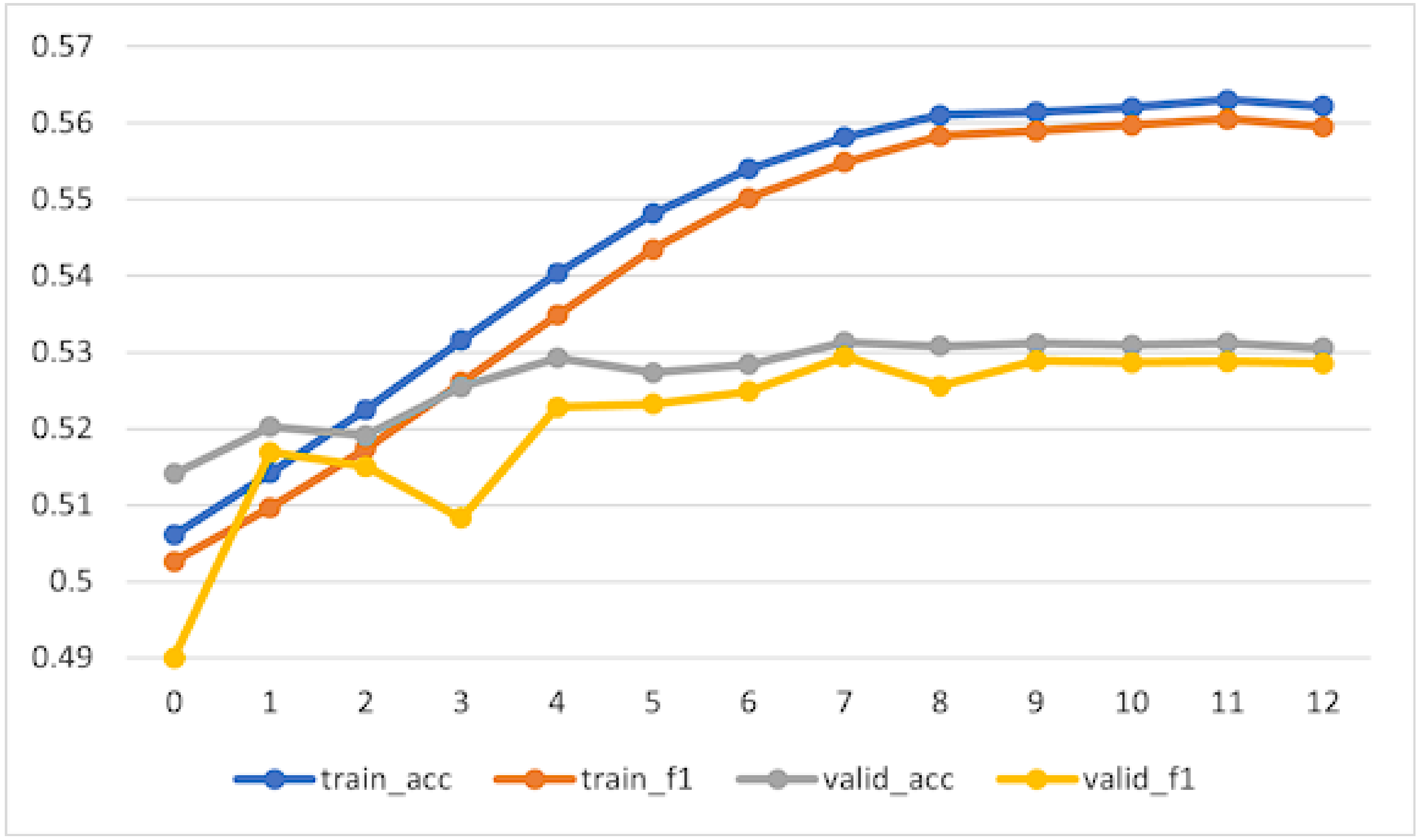


	Our Design	**Original Paper**
Learning rate	1e-4, with Cosine decay function	1e-5
Optimizer	AdamW, with 0.001 weight decay	Adam
Train-Valid Split	stratified 3-fold splitting, each fold take 33.3% data as validation set	randomly select 30% of training data as validation, rest 70% to train model
Model Stacking	each model do the 3-fold training and average the 3 sub-models results to predict	Each CNN model retrained 5 times and do average forecast
Label	- binary classification of future 20-day return; - binary classification of future 5-day return; - three-classification of 20-day return(2 for NA return)	binary classification of future 20-day return –1 positive, 2 negative

Training Results



(a) loss



(b) acc & f1 score

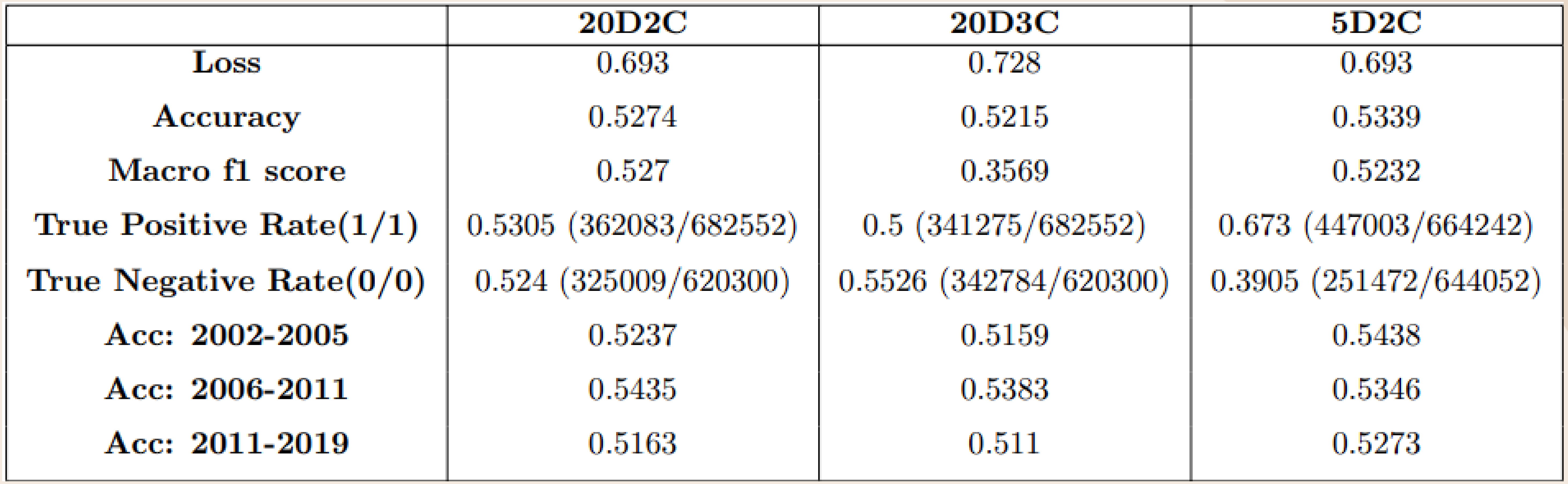
Based on the model architecture mentioned above, our final model boasts a total of **1,417,732** parameters. It requires 16 minutes to train a single epoch on the Kaggle P100 GPU and moreover 12 hours for the entire 3-fold modeling workflow.

Here we use the 20D2C(2 classification label of future 20-days return) model as an example. Figure 4 demonstrates the changing of the **training & validation loss** over the epochs, as well as its accuracy and macro f1 score on the valid set. It is clear that the model is close to convergence after 8, 9 epochs.

3

Model Performance

Testing Metrics



	20D2C	20D3C	5D2C
Loss	0.693	0.728	0.693
Accuracy	0.5274	0.5215	0.5339
Macro f1 score	0.527	0.3569	0.5232
True Positive Rate(1/1)	0.5305 (362083/682552)	0.5 (341275/682552)	0.673 (447003/664242)
True Negative Rate(0/0)	0.524 (325009/620300)	0.5526 (342784/620300)	0.3905 (251472/644052)
Acc: 2002-2005	0.5237	0.5159	0.5438
Acc: 2006-2011	0.5435	0.5383	0.5346
Acc: 2011-2019	0.5163	0.511	0.5273

To assess the model's performance on the test set, we predicted the three models on the test data and evaluated the results using **ACC, F1_SCORE**, and other metrics. The metrics results are shown in Table.

Our findings indicate that the addition of **NA values** to the labels does not significantly impact the accuracy of the model.

However, the model utilizing future 5-day earnings as labels appears to exhibit **higher accuracy**.

Furthermore, we examined the model's performance across different years and observed a **notable decline** in accuracy for the time period far removed from the training set (2011-2019). This may suggests that the model's predictive power **diminishes over time**, indicating limited long-term forecasting capabilities.

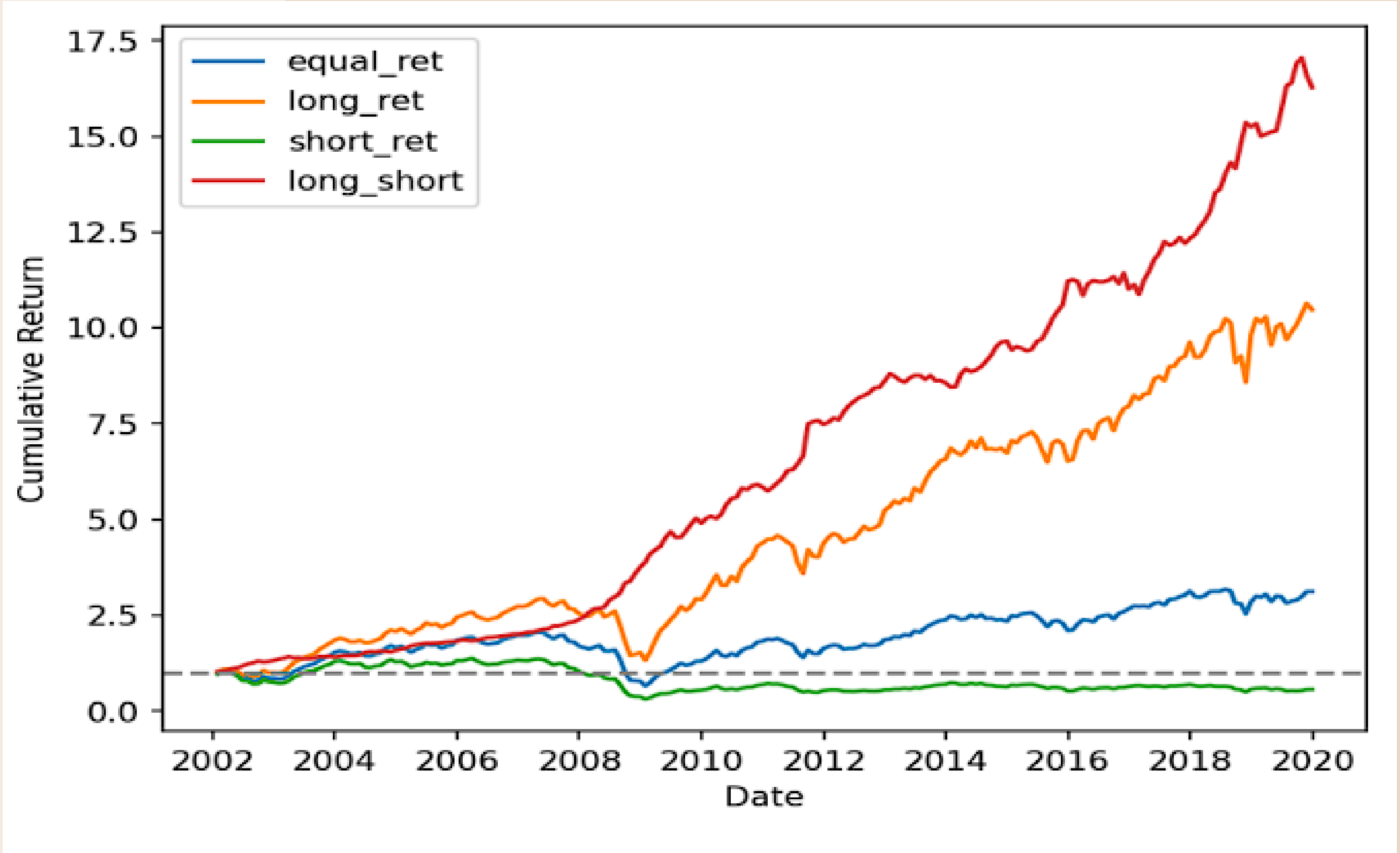
Back-testing Results

In addition to model evaluation metrics such as the acc and f1 score, we build the stock portfolio based on the model prediction signal.

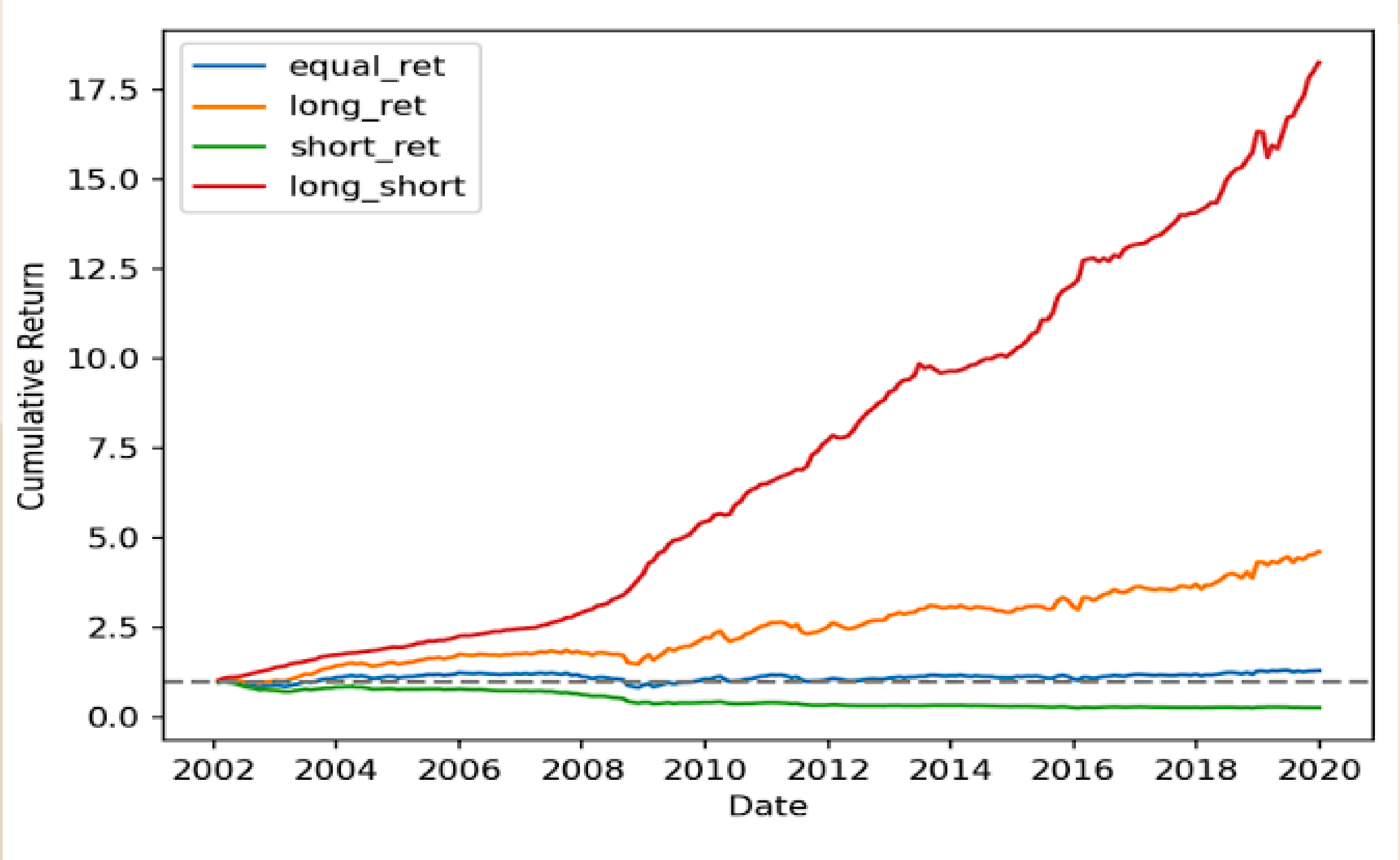
For each timestamp, we select the **top 10% stocks** with the strongest **uptick signal as the long** portfolio(orange line), while the **short** portfolio comprises the top 10% stocks with the strongest **downward signal**(green line). Finally, the returns of the model **long-short portfolio**(red line) and the market average portfolio(blue line) were compared.

As evident from the figure, the model-based portfolio exhibits **significant return ability in the long run**. However, it appears that the majority of the returns **come from the long end**, indicating that the model's **short-selling prediction ability is relatively weak**.

This suggests that there may be room for improvement in the model's short-side predictions, which could potentially enhance the overall performance of the portfolio.



(a) CNN20D2C



(b) CNN5D2C

4

Conclusion & Future Work

Conclusion

It's clearly that CNN network **does have** some ability to learn from the stock's K-line plot and give some plausible predictions on future return. Although there is obviously ample room for further improvement of the model's accuracy, this paper has provided us with valuable inspiration on **how to apply image models to time-series data** problem. Due to the time limitation and the restrained computing resources, we can not fully exploit our research on this direction. However, we have outlined some potential challenges and avenues for future exploration in this paper.

Model Design

- This paper's architecture is basically derived from the VGG model. This design makes it harder for the network to go deeper. Therefore, we suggest try some ResNet type architecture(like RepVGG) and multi-parallel channel structure to build the model, which may enhance the model ability.

- Our replication model‘s total parameters is about 140k, and the training data-set’s length is only 885k. The low parameter-to-sample ratio can probably cause the model over-fitting, which may partially explain the poor performance on the test data-set.

Future Work

Label Design

- Only using future return as an indicator may not be a good choice for model's learning. Because the stock's future price may be go up intensively and then falls rapidly. In this scenario, although the future return may ends up positive, it's hard to say is good to holding this stock right now. Therefore, simply use future return to construct classification label may not be a good choice for model's learning and it may even confuse the model. Thus, we may take the risk and max draw down into consideration in future label design.

Loss Design

- Weighted Loss: In a three-class classification model, we are usually more concerned about the error of predicting the exact opposite signal than the error of predicting NA. Therefore, in the design of the loss function, we may need to assign different weights to each sample for prediction and calculate the weighted loss. This may help our model reduce the possibility of predicting completely opposite results.

Training Design

- Periodical Recursive training may be helpful to the future prediction since the model can learn some recent information.
- Transfer learning: fit the model with Chinese A share market data and check the model's sustainability.

Question Time

THANK YOU SO MUCH!