Paper replication:

(Re-)Imag(in)ing Price Trends

Li Chenghai(20828446): Coding and Report Writing
Liu Shifei(20734530): Coding Support and Report Writing
Nie Jialei(20747874): Coding Support and Report Writing
Wu Jiajun(20666111): Coding Support and Report Writing

Nov. 13, 2021

Ⅰ. Introduction

This paper explores convolutional neural networks that flexibly learn price patterns as images that are most predictive of future returns. The raw predictor data are images - stock-level price charts, from which authors model the predictive association between images and future returns using a convolutional neural network (CNN).

Our main task is to learn and replicate the core content of the paper, including data preparation, model design, performance evaluation, etc.

Ⅱ. Data Description

A data is a black and white image, and each image is composed of 64*60 data points which are either 0 or 255, representing black or white. We used first seven-year (1993-1999) sample data as training and validation data, in which 70% of the sample data are randomly selected for training and the remaining 30% for validation, a total of 67858 data. The remaining twenty years of data (2000-2019) comprise the out-of-sample testing data.

The following picture shows the true value of x.

```
In [4]: print(images.shape)
        images[0]

        (67858, 64, 60)

Out[4]: 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)
```

The following picture shows the true value of y.

Out[5]:

	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

Ⅲ. Methodology

Firstly, we imported the packages that need to be used, and set the seed for the pytorch to generate random numbers. After that, we downloaded the data and previewed the data. We saved the csv file in feather format in order to greatly speed up the reading speed, and judged whether the number of images and labels are equal, if they are not equal, there will be an error, and if they are equal, the process will proceed smoothly.

Thirdly, we built datasets and defined img, label and len. The stock returns were classified into two categories depending on whether the return is positive or negative. In this case, we chose batch_size equal to 32.

Next, we showed the shape layer by layer. We chose one image and reshaped it. In addition, padding was equal to ((kernel_size - 1) * stride * dilation) / 2. The following picture shows that one image is gradually reduced to a vector of only two elements.

```
torch.Size([64, 60])
torch.Size([1, 64, 27, 60])
torch.Size([1, 64, 13, 60])
torch.Size([1, 128, 10, 60])
torch.Size([1, 128, 5, 60])
torch.Size([1, 256, 7, 60])
torch.Size([1, 256, 3, 60])
torch.Size([46080])
torch.Size([2])
```

Next, we built the neural network and executed the four functions Conv2d, BatchNorm2d, LeakyReLU and MaxPool2d in sequence. Each group of four of them together was called one unit. Convolutional layer and max pooling layer could change the shape of x. At each convolutional layer we were able to set the number of output channels manually. Max pooling layer would decrease the shape of x by some factor. Batch normalization layer and leaky ReLu function could not change shape of x but only modified the contents. After these units of convolutional layers, the number of channels of x increases from 1 to 256, the height decreases from 64 to 3, and the width remains the same at 60. Then x was flattened and reshaped into a one-dimensional vector. We applied a fully connected layer and a Softmax activation function to receive a probability tuple of binary classification.

Then we used built neural network model to have a look and test it briefly. Finally, we carried out the training process. We defined loss as the gap between the predicted value and the true value. Function loss.backward() was applied to calculate gradients, which were the derivatives of the loss with respect to parameters. Function optimizer.step() was used to correct the weight parameters of the neural network based on the gradients. When all the data were put into training for once, it was called one epoch. We stipulated that the whole process was repeated ten times, thus epochs = 10.

Gaussian Error Linear Unit

The Gaussian Error Linear Unit, or GELU, is an activation function. The GELU activation function is $x\Phi(x)$, where $\Phi(x)$ is the standard Gaussian cumulative distribution function. The GELU nonlinearity weights inputs by their percentile, rather than gates inputs by their sign as in ReLUs. Consequently, the GELU can be thought of as a smoother ReLU.

Layer Normalization

Unlike batch normalization, Layer Normalization directly estimates the normalization statistics from the summed inputs to the neurons within a hidden layer so the normalization does not introduce any new dependencies between training cases. It works well for RNNs and improves both the training time and the generalization performance of several existing RNN models. More recently, it has been used with Transformer models.

We compute the layer normalization statistics over all the hidden units in the same layer as follows:

$$\mu^l = \frac{1}{H}\sum_{i=1}^{H} a_i^l$$

$$\sigma^l = \sqrt{\frac{1}{H}\sum_{i=1}^{H}(a_i^l - \mu^l)^2}$$

where H denotes the number of hidden units in a layer. Under layer normalization, all the hidden units in a layer share the same normalization terms μ and σ, but different training cases have different normalization terms. Unlike batch normalization, layer normalization does not impose any constraint on the size of the mini-batch and it can be used in the pure online regime with batch size 1.

Xavier Initialization

Xavier Initialization, or Glorot Initialization, is an initialization scheme for neural networks. Biases are initialized be 0 and the weights W_{ij} at each layer are initialized as: $W_{ij} \sim U[-\frac{1}{\sqrt{n}}, \frac{1}{\sqrt{n}}]$, where U is a uniform distribution and n is the size of the previous layer (number of columns in W).

Softmax

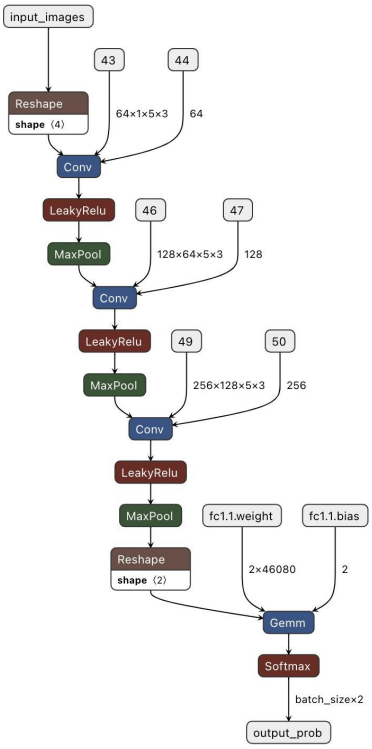
The Softmax output function transforms a previous layer's output into a vector of probabilities. It is commonly used for multiclass classification. Given an input vector x and a weighting vector w we have:

$$P(y = j \mid x) = \frac{e^{x^T w_j}}{\sum_{k=1}^{K} e^{x^T w_k}}$$

Loss function

We used cross entropy loss.

$$l_n = -w_{y_n} \log \frac{exp(x_{n,y_n})}{\sum_{c=1}^{C} exp(x_{n,c})}$$



Ⅳ. Prediction

We used CNN to downsize the image into a vector with two elements. The first element represents the probability of price going down and the second element represents the probability of price going up. Then we compared two elements and chose the great probability to decide whether the price will go up or down.

```
y_pred[0:6,]
```

```
array([[0.68374044, 0.3162595 ],
       [0.609837  , 0.39016303],
       [0.4750822 , 0.5249178 ],
       [0.665544  , 0.334456  ],
       [0.5044233 , 0.4955767 ],
       [0.5820372 , 0.41796282]], dtype=float32)
```

We used four different CNN models to do the predictions, including Baseline, GELU, GELU+LN, LN. The baseline model used Leaky ReLU, BN and Xavier. The setup was Random Seed 42, lr 1e-5, Batch size 128 and Adam Optimizer. The models could be exported as onnx format for cross-platform training. We calculated the loss value, accuracy rate, Sharpe ratio and correlation between probability of price going up and return. Among these four models, we found the LN model was the best one. The checkpoints of models could be found in the source folder. The following table is the calculated results.

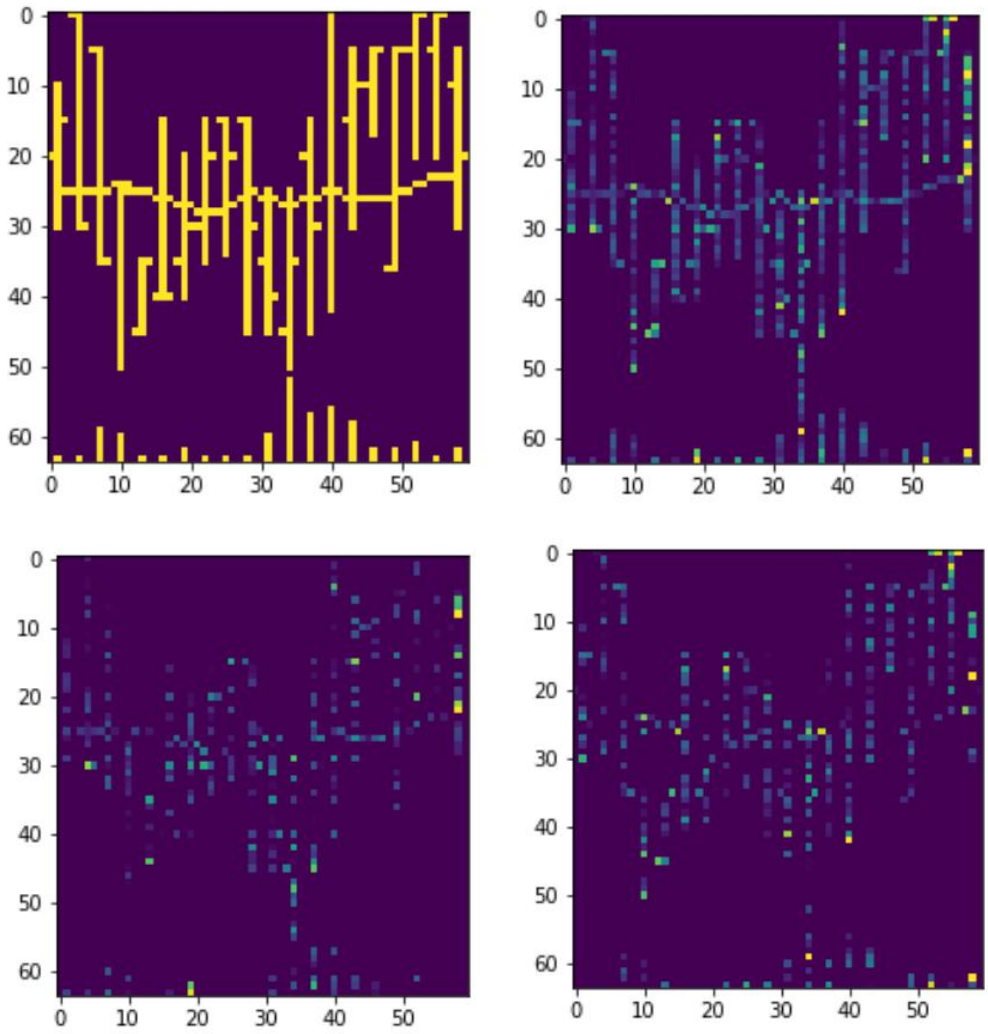
	Loss			Acc.	Correlation		Sharpe Ratio
	Train	Val	Test	Test	Pearson	Spearman	Test
Baseline	0.688653	0.686537	0.693601	0.526262	0.002834	0.004692	0.136584
GELU	0.686015	0.686316	0.693599	0.524919	0.003045	0.004661	0.050687
GELU+LN	0.681694	0.685956	0.692950	0.526351	**0.003220**	**0.004933**	0.142084
LN	**0.680397**	**0.685894**	**0.692381**	**0.528553**	0.002920	0.004514	**0.504030**

Ⅴ. Extension-Interpretation

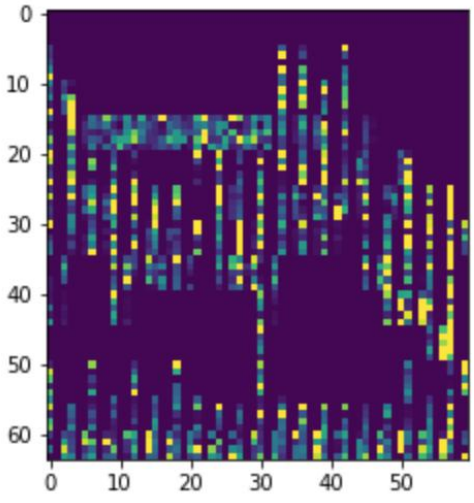
The first picture is a sample of the original image, which is a OHLC chart of 20-day stock price. For each candle, the top and bottom represent the high and low price. The bar on the left represents the open price, while the bar on the right represents the close price. If the bar on the left is higher than the bar on the right, there is a price drop.

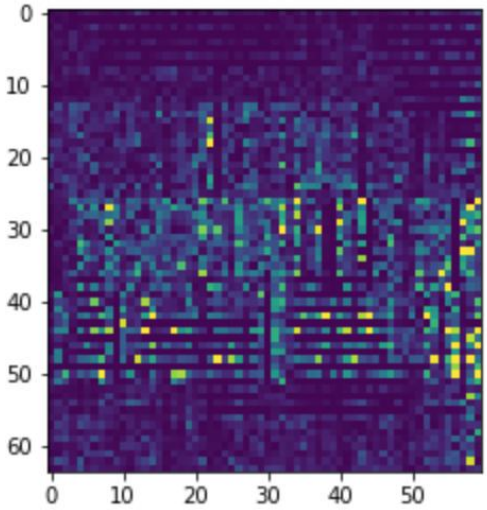
The second picture shows the first derivative of the predicted probability of price growing with respect to each data point on image. The brighter the point is, higher sensitivity it has. This picture can show which point have greater impact on the prediction.

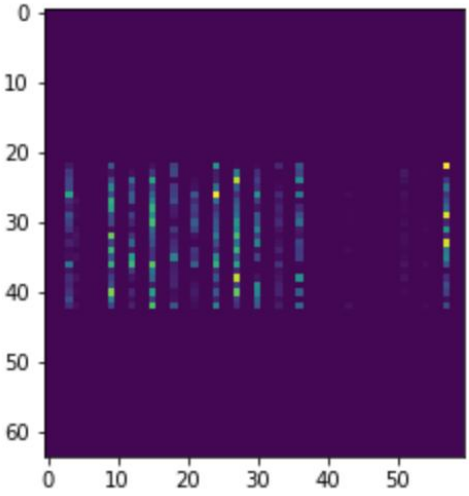
The third and fourth picture separate the first derivative into positive and negative value. We can see which data points are related to price growing and which are related to price decline.



Another approach for interpretation is called Grad-CAM, Gradient-weighted Class Activation Mapping. It is a visualization technique. Gradients of the probability of final classification flowing into different convolutional layers are used to produce a heat map highlighting the important regions in the image for the prediction. For the first layer, the neural network is intensively activated when the volatility is high. This is shown in the lower right corner of the picture, where for the last several days the difference between high and low prices is large and these bars are brighter compared to others. For the second layer, we see that open and close prices tend to be bright points, which means that they are emphasized by the model. For the third layer, the neural network is activated when the volume is high. It is shown in the middle and last days of the picture, where the volume is higher and at the same time the pixel points are brighter than the rest.


0
10
20
30
40
50
60
0
10
20
30
40
50


0
10
20
30
40
50
60
0
10
20
30
40
50


0
10
20
30
40
50
60
0
10
20
30
40
50