Paper Replication: (Re-)Imag(in)ing Price Trends

Bo Huang, Mingyang Chen[*]
Data Science and Analytics Thrust
The Hong Kong University of Science and Technology
{bhuangas, mchenbt}@connect.ust.hk

Abstract

Due to the rapid growth of financial markets, research related to stock prediction has received a lot of attention. The acquisition and processing of stock data, as well as the selection and training of models, are very critical. Most prior approaches convert stock data into a 1D time-series data for analysis, which is not a form of data that both humans and convolutional neural networks are good at handling. In this paper, we re-explore the feasibility and performance of converting stock data into image data to train the return predictor. First, we convert stock data into image form by plotting stock trends and adding some important graphical description components, and take advantage of the powerful perception and extraction capabilities of CNNs for image data to train CNN-based stock predictors and conduct extensive experiments for comprehensive evaluation. In addition, since self-supervised learning and Masked Autoencoders (MAE) break through previous performance bottlenecks in image recognition recently, we also investigate the use of MAE for pre-training of stock image data and finetuning for downstream stock prediction tasks. Our replication work can further demonstrate the results that can be achieved by imaging stock data as well as give some implementation details. Also, the initial attempt of MAE can deepen our understanding of this prediction task and provide reference for applying MAE to other related tasks.

1 Introduction

As financial markets continue to grow in size and investor demand for market information continues to increase, information has become one of the most important factors affecting portfolio performance. Investors receive a variety of different kinds of information every day, including a large number of historical numbers, as well as will be exposed to an increasing number of new facts, data and statistics. Therefore, how to effectively analyze and process this information is of great importance to improve the efficiency of investment decisions. To assess the predictability of stock returns, it is necessary to analyze them in terms of the behavioral characteristics of investors. One of the most important factors in stock prediction is the relationship between historic stock data and future returns. Traditional analysis methods are mainly through statistical analysis of data, but due to the time required and the limitations of the analysis methods, the use of machine learning techniques can better compensate for these shortcomings.

[*]This work is primarily constituted of two parts, which are jointly accomplished by the two authors. For the first part, Mingyang Chen reproduces the framework of using CNN to predict the return trend of 2-D stock data. Specifically, Mingyang reconstructs the baseline models introduced in the original paper and verifies the performance, robustness and interpretability of the model from several aspects. For the second part, Bo Huang explores the feasibility of replacing the CNN-based feature extractor by adapting Masked Autoencoder (MAE) and self-supervised learning in this task. Specifically, Bo explicitly introduces the adaptation process and studies the influence of mask ratio on the prediction performance. Finally, the paperwork is organized and compiled by the two authors equally.

Machine learning techniques have been widely used in the field of finance. In stock prediction, it is important to mine the knowledge implied by historical stock data. This is manifested in how to select a training model and evaluate its performance reasonably as well as how to construct a suitable large number of training samples for the training of the model. For the selection of training models, two different machine learning algorithms mainly exist: nonparametric estimation and parametric estimation. Non-parametric methods mainly face the so-called "curse of dimensionality" problem, while parametric methods also have many trade-offs in terms of function form and scale, and the phenomenon of data overfitting can occur as the number of estimated parameters increases.

Leveraging machine learning models to make stock prediction is essentially a supervised learning scheme. It refers to given the historical stock data containing features and their corresponding labels, and then a suitable machine learning model is used to build up a mapping between the features and labels. In the real world, where stock data is often very cumbersome, one of the most important tasks in stock prediction is data collection and processing. Typically, stock data is collected through daily value data tables that record various indicators of the stock. In addition to time information, there are many other aspects of data, such as corporate governance, profitability, *etc*., but these data are not directly available for investment analysis. Obviously, since stock data involves many indicators and is updated on a daily basis, stock data is very voluminous and is a time-series type of data. A typical dataset for stock forecasting is a 1D series dataset based on data tables, which is inherently noisy and incomplete, and thus requires heavy data pre-processing before it can be used for training models.

Actually, it should be noted that even with well-processed stock data in the form of time series, it is difficult for human experts to directly obtain valuable information from these data for stock prediction, which is caused mainly by the form of the data because humans are not good at extracting information from 1-D time series data. However, humans are highly capable of observing and processing image data. For example, an experienced stock investor can quickly get useful information from the graphical movements of a stock.

Inspired by that, [10] proposed to design embeds 1D time series data in a higher dimensional space, representing it as a 2D image, before bringing it to predictive analysis. There are two main motivations. The first one is that leading CNNs (Convolution Neural Networks) architectures are tailored for image analysis. CNNs are extensively applied in a wide range of computer vision tasks and are capable of achieving even better than human-level performance, and it is useful to represent stock data in the format that CNNs naturally ingest (*i.e*., images). Second, representing time series as an image allows the model to focus on relational attributes of the data that would be difficult to tease out with time series methods. It is the same basic logic for why humans illustrate patterns graphically rather than with lists of numbers.

On the other hand, the training of human-outperforming models relies heavily on a large amount of labeled data, which are very expensive to obtain. To lift the restriction on the requirement of labeled data, self-supervised pre-training is widely used in many NLP-related tasks. Among these self-supervised pre-training approaches, GPT [2] based on autoregressive language and BERT [5] based on masked autoencoding are straightforward and conceptually simple, the core mechanism behind the two methods is to guide the model to predict what is removed from the input before it is passed to the model. These kinds of strategies have achieved great success in many fields of NLP and make it feasible to train well-performing NLP models containing a huge number parameters. At the same time, [14] reveals that masked autoencoders can be natural and applicable in computer vision as well, whose architecture highly depends on Vision Transformers (ViT) [6]. Generally speaking, ViT is a form of transformer for computer vision, have been introduced to bridge the gap that Convolution Networks (CNNs) cannot directly integrate mask tokens [5] or positional embedding [13] into model training.

2 Using CNN to Learn "Imaging" Market Data

2.1 Introduction to the CNN-based Return Trend Classifier

After transforming the 1-dimensional time series data to 2-dimensional integrated image data of OHLC chart with average lines and volume bars, we can employ machine learning models to try to capture the complex and subtle patterns implied in different horizontal return figures by mimicking the human-perception. And this task is considered to be naturally suitable for Convolutional Neural Networks (CNNs) [12, 9]. A typical CNN is constituted of two parts including a CNN-based feature extractor and a Multilayer Perceptrons (MLP)-based classifier. Compared to the traditional neural network which is only constituted by concatenating several fully-connected (MLP) layers, a CNN can deal with very high-dimensional hierarchical data, e.g., images, by learning filters (or kernels) automatically. In traditional algorithms, these filters are hand-engineered. And for CNNs, the convolutional layers perform dot products of the convolutional kernel with the input matrix of the layer and feedforward the outputs for the following part of the model. Subsequently, convolutional kernels are optimized through each back-propagation automatically. A common CNN block is generally constituted by a convolutional layer (abbreviated as conv layer), an activation function, e.g., ReLu[1], and a pooling layer for reducing the dimensions of input by combining the outputs of neuron clusters at one layer into a single neuron. The following equation can be used to calculate the **height** and **witdh** of a feature map output by a conv layer or a pooling layer:

$$\begin{aligned} H_{out} &= \left\lfloor \frac{H_{in} + 2 \times \mathit{padding}[0] - \mathit{dilation}[0] \times (\mathit{kernel\_size}[0] - 1) - 1}{\mathit{stride}[0]} + 1 \right\rfloor \\ W_{out} &= \left\lfloor \frac{W_{in} + 2 \times \mathit{padding}[1] - \mathit{dilation}[1] \times (\mathit{kernel\_size}[1] - 1) - 1}{\mathit{stride}[1]} + 1 \right\rfloor \end{aligned} \tag{2.1}$$

In [10], the authors proposed a baseline model of a binary CNN-based classifier to predict the decisions of return trends of different horizons including 5-day, 20-day and 60-day. For the specific setup, they state to construct n conv blocks and every block uses 5×3 conv filters and 2×1 max-pooling filters for convenience. For predicting 5-day, 20-day and 60-day return trends, they set n to 2, 3 and 4 respectively. The corresponding depths of the outputs of different blocks are set to 64, 128, 256 and 512 in a progressive order. Besides, the authors also set the vertical and horizontal strides of conv layers as (3,1), and set the vertical and horizontal dilations as (2,1). However, according to Equation (2.1), the height of feature map output after the third layer (including the third layer) will be a negative number under the proposed setup in our calculation. In our practice, the program will also raise an error when the input is a 64×60 20-day sample image. We suppose the authors also add some paddings to the input feature map of each block during the forward propagation. However, because the authors did not explicitly introduce the hyperparameters for padding, we also set padding to zero in our models. And to make the model can be well constructed as the authors proposed, we use the identical architecture as [10] but set vertical and horizontal strides to (1,1) and fix the block number to 3. Our implemented model is illustrated by Figure 1. Besides, we also follow the design of [10] to employ Batch Normalization [9], Xavier Initialization [7] and Leaky ReLu [3] in our implementation. Moreover, This model is set as the baseline for testing the variation robustness of model architecture in Section 3.

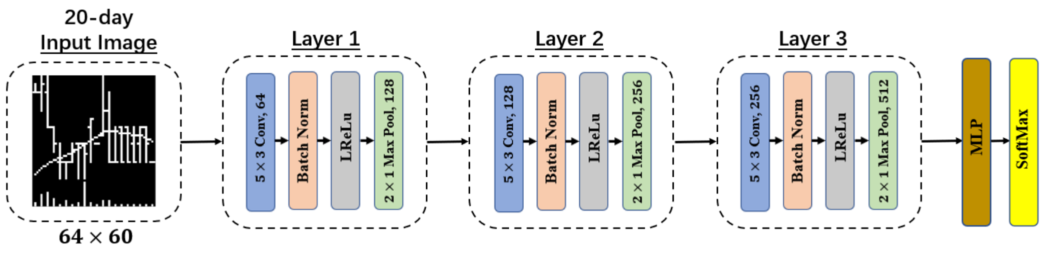


Figure 1: Overview for baseline model of CNN-based return trend classifier.

2.2 Classification Accuracy and Correlation of Predicting Return Trend

In this section, we test the feasibility of using CNN-based classifier to predict the return trend for different horizons, e.g., 5-day, 20-day and 60-day. In Table 1, we report the reproduced results collected by our implemented model, which include predicting accuracies and correlations, of Table 2 in [10]. We use the following equation to calculate the classification accuracy:

$$\mathit{Accuracy} = (TP + TN)/(TP + TN + FP + FN) \tag{2.2}$$

Where, TP, TN, FP and FN refer to the terms of True Positive, True Negative, False Positive and False Negative respectively. And according to the original setup, A TP or TN occurs when a predicted "up" probability of greater than 50% coincides with a positive realized return and a probability less than 50% coincides with a negative return. FP and FN are the complementary outcomes. Besides, because the definition of correlation metric is not explicitly introduced by the authors, we propose to use the *Pearson Correlation* which is used in Table 18 of [10]. Here is the specific formulation of the equation:

$$corr_{xy}^{pe} = \frac{\sum_{i=1}^{n}\left(x_i-\bar{x}\right)\left(y_i-\bar{y}\right)}{\sqrt{\sum_{i=1}^{n}\left(x_i-\bar{x}\right)^2}\sqrt{\sum_{i=1}^{n}\left(y_i-\bar{y}\right)^2}} \tag{2.3}$$

Where, x and y refer to the hard label returned by classifier and the ture label respectively in our calculations. In Table 1, we use the same test set as [10], which is 1,403,975 stock data including return values and return labels collected from the year 2001 to the year 2019. And we also randomly shuffle the stock data collected from the year of 1993 to the year 2000 and selected 70% data to constitute the training dataset and 30% data for validation. By comparing the results collected by our trained model with the results reported in Table 2 of [10], our test accuracies are near 1% lower. We are not sure whether this phenomenon is caused by the information which is not explicitly reported by the author, e.g., the vertical dimension problem of the feature map raised by the original architecture setup which we reported above. Moreover, we also find that the validation and test accuracies are highly influenced by the selection of training data in our random split test.

Table 1: Classification Accuracy and Correlation Collected on *Validation Set* and *Test Set*.

Return Horizon	Accuracy		Correlation	
	Val	Test	Val	Test
05-Day	0.555	0.523	0.111	0.044
20-Day	0.526	0.521	0.056	0.033
60-Day	0.528	0.516	0.057	0.039

2.3 Exploring the Relationship between Training and Validation Stages of Model

In [10], the authors state that a early stop condition is needed for keeping learned models away from over-fitting or noneffective training. For this purpose, the authors proposed to stop the model training when the validation loss no longer drops in the past two epochs. Although the authors do not explicitly explore the tradeoff between training and validation stages, we find that the training loss is big and the training accuracy is still low when the early stop condition is triggered (within 10 epochs in general). To prevent the model suffer from the under-fitting problem and to explore the relationship between the training and validation Stages of models, we force the model to be trained for 200 epochs among three prediction tasks, e.g., return trends of 5-day, 20-day and 60-day. We set the batch size as 128 and the learning rate as 1e-5, which is identical to the setup of the original paper. In Figure 2, we plot the comparisons between training and validation losses in the first row and plot the comparisons between training and validation accuracies in the second row. We can see that the validation loss only continually drops in the very beginning stage, and the increment of validation accuracies also follows this trend. These observations demonstrated that the early stop condition can effectively prevent the model from being overfitting.

3 Exploring the Sensitivity and Interpretability of the Model

In this section, we first test the sensitivity of model predictions among different classification thresholds. Secondly, to verify the model robustness against architecture variations, we report the test performance of several varietal models derived from the baseline model reported in Section 2. Finally, to explore the specific patterns in the historical stock data learned by the model that affect forecast results the most, we employ the Grad-CAM [11] method to plot the activation map w.r.t each CNN layers of the model.

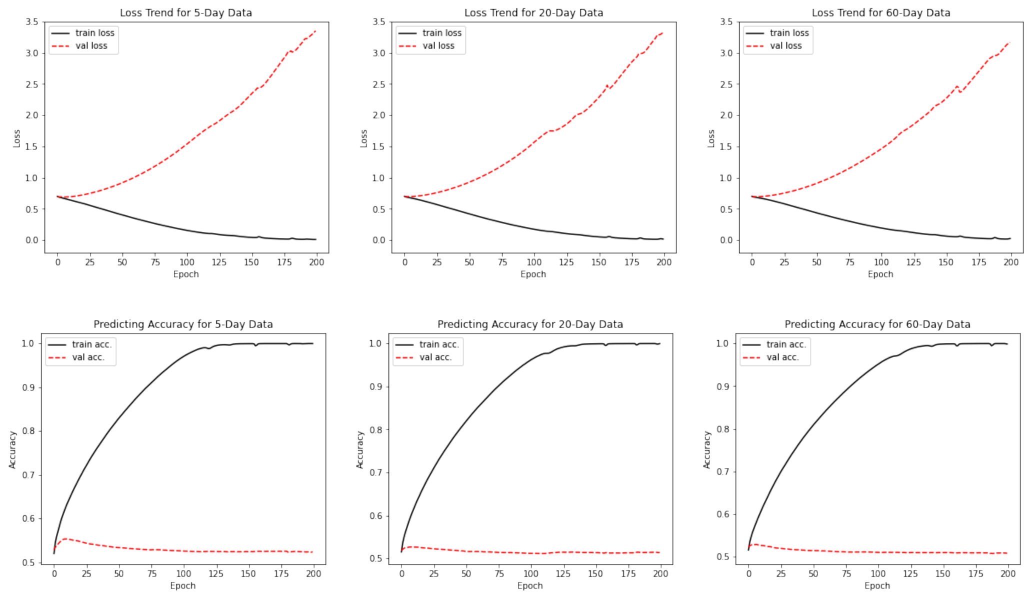
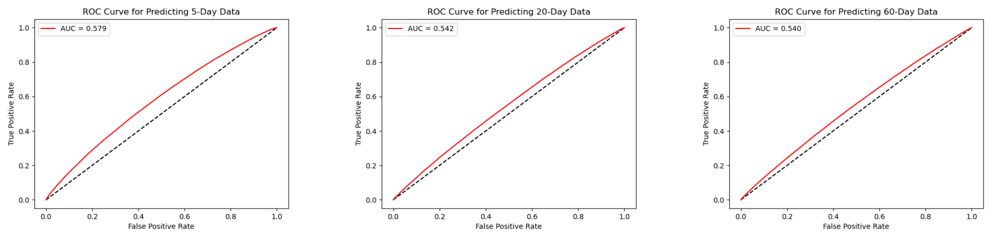


Figure 2: The first row demonstrates the comparisons between training and validation losses among three predicting tasks including the return trends of 5-day, 20-day and 60-day. The second row demonstrates the corresponding comparisons between training and validation accuracies.

3.1 Using ROC Curve to Test the Sensitivity of Model Classification

In our implementation, we observed that the ratio between the number of positive and negative samples in the test data is close to 1.08, which is not balanced enough. To evaluate the model performances unbiasedly, and to explore whether the current classification threshold, which is set to 0.5, is reasonable, we draw ROC curves for the prediction results. Defining False Positive Ratio (FPR) as the horizontal axis and True Positive Ratio (TPR) as the vertical axis, a ROC curve can be drawn by linking the values of TPR and FPR collected under different classification thresholds. Where, the TPR and FPR can be calculated by $TPR = TP/(TP + FN)$ and $FPR = FP/(FP + TN)$. We plot the ROC curves for predicting return trends among 3 different horizons in Figure 3. We can see that the three models generally achieve the best tradeoff when the threshold is around 0.5.



(a) 5-Day (b) 20-Day (c) 60-Day

Figure 3: ROC curves for predicting return trends among 3 different horizons.

3.2 Verifying the Sensitivity to Model Structure and Estimation

In this section, we explore the sensitivity of the choice of model structure to various metrics in multiple dimensions. Among them, the metrics we focus on include loss function value, classification accuracy, average cross-sectional correlation of forecasts with return realizations (both Pearson and Spearman rank correlation), and equal weighted (EW) Sharpe Ratio. Follow the experimental setup of Table 18 in [10], for loss and accuracy, we report statistics for both the validation (Val) and test

(Test) samples. Correlations and Sharpe ratios are for the test sample only. Because the authors do not explicitly introduce the specific definition of Pearson and Spearman rank correlation as well as the annual Sharpe Ratio, we will use the Equation 2.3 to calculate Pearson correlation and use the following equation to calculate the Spearman correlation:

$$corr_{xy}^{sp} = \frac{\sum_{i=1}^{n}\left(R(X)_i - R(\bar{X})\right)\left(R(Y)_i - R(\bar{Y})\right)}{\sqrt{\sum_{i=1}^{n}\left(R(X)_i - R(\bar{X})\right)^2}\sqrt{\sum_{i=1}^{n}\left(R(Y)_i - R(\bar{Y})\right)^2}} \tag{3.1}$$

Where $R(X)$ and $R(Y)$ are two ranks of X and Y. In our case, X and Y refer to predicted labels and true labels, respectively. We calculate these two correlations by calling the *pearsonr* and *spearmanr* functions of SciPy[2]. And we also use the following equation to calculate the Sharpe Ratio:

$$SR_X = \frac{\mu_X - R_f}{\sigma_X} \tag{3.2}$$

Where, μ_X and σ_X refer to the mean and standard deviation of predicted hard labels of test data. And R_f refers to a weight to calculate the Sharpe Ratio, and we set it as 0 in our implementation. In Table 2, we report the sentivities of different metircs against varying model structure in multiple dimensions. It should be aware that the empty rows with "-" symbols refer to the replacements that will cause a negative vertical dimension problem of feature map output by the final (the third) CNN block, which we have raised in Section 2.
Back onto the track, by observing the collected results, we can verify that model predictions are robust against partially varying the structure of the model. Furthermore, we also find another interesting observation that test performances became better when we close the batch normalization in our task. This observation is also partially revealed by the data reported by Table 18 of [10]. We suppose that this may because the information contained in the constructed 2-D OHLC chart images is high-frequency and dense, which leads to a large difference between the data, and subsequently leads to a significant distribution gap between the training data and the test data. This maybe the reason why we get a relatively poor result when processing the data at the test stage with the mean and standard deviation learned from the training stage. This may also be the reason why the model in figure 2 overfits at a very early stage during training.

Table 2: Sensitivity to Model Structure and Estimation for Predicting 20-day Return Horizon Trend.

Baseline	Variation	Loss		Accuracy		Correlation		Sharpe Ratio
		Val	Test	Val	Test	Spearman	Pearson	
Baseline	none	0.696	0.699	0.526	0.521	0.033	0.033	1.322
Filter (64)	32	0.696	0.699	0.525	0.521	0.032	0.032	1.403
	128	0.694	0.698	0.528	0.522	0.035	0.035	1.338
Layers (3)	2	0.694	0.701	0.534	0.517	0.035	0.035	0.974
	4	-	-	-	-	-	-	-
Dropout (0.5)	0	0.695	0.698	0.526	0.521	0.034	0.034	1.331
	0.25	0.695	0.698	0.525	0.521	0.034	0.033	1.371
	0.75	0.695	0.699	0.526	0.521	0.032	0.032	1.352
BN (yes)	no	0.686	0.693	0.546	0.526	0.053	0.053	0.987
Xavier (yes)	no	0.694	0.698	0.527	0.522	0.034	0.034	1.379
Activation (LReLU)	ReLu	0.696	0.699	0.523	0.521	0.032	0.032	1.402
Max-Pool Size (2×1)	(2×2)	0.695	0.698	0.520	0.514	0.026	0.026	1.029
Filter Size (5×3)	(3×3)	0.695	0.704	0.528	0.513	0.031	0.031	0.915
	(7×3)	-	-	-	-	-	-	-
Dilation (2×1)	(1×1)	0.695	0.703	0.527	0.514	0.031	0.031	0.906
Stride (1×1)	(3×1)	-	-	-	-	-	-	-

3.3 Visualizing and Interpreting the Patterns Captured by the Model

Despite the improvement of model performance in predicting the return trend, we are also curious about what are the key features of an effective machine model paying attention. Moreover, we also want to explore the feasibility of predicting the return trend by jointly referencing the structured human knowledge and the nonrepresentational model attentions. To visualize the model's attention

[2]https://scipy.org/

to the input when it gives the "up" or "down" decision, we employ Grad-CAM [11], which can produces heatmaps for each layer of the CNN that illustrate the regions of the input most important for predicting a class. When plotted for individual observations, these activation heatmaps help describe which aspects of the image are the most important determinants of the CNN's prediction. In Figure 4, we mainly plot the activation heatmaps for each three CNN blocks when predicting "up" or "down" labels by our implemented model. To collect higher-quality heatmaps to help us interpret the model decisions, the samples shown in figure 3 are all selected from the samples which are correctly classified by the model with high logit scores. According to the observation, we find that the shallow layers, e.g., the first and the second CNN blocks, mainly pay more attention to the open and close prices, which are especially illustrated by the bright points. Moreover, we can see that the deeper layer such as the third CNN block always pays more attention to the bars of price fluctuation. Another interesting finding is that When there are multiple large price fluctuation bars in the input, the model pays more attention to the moving average line rather than other information.

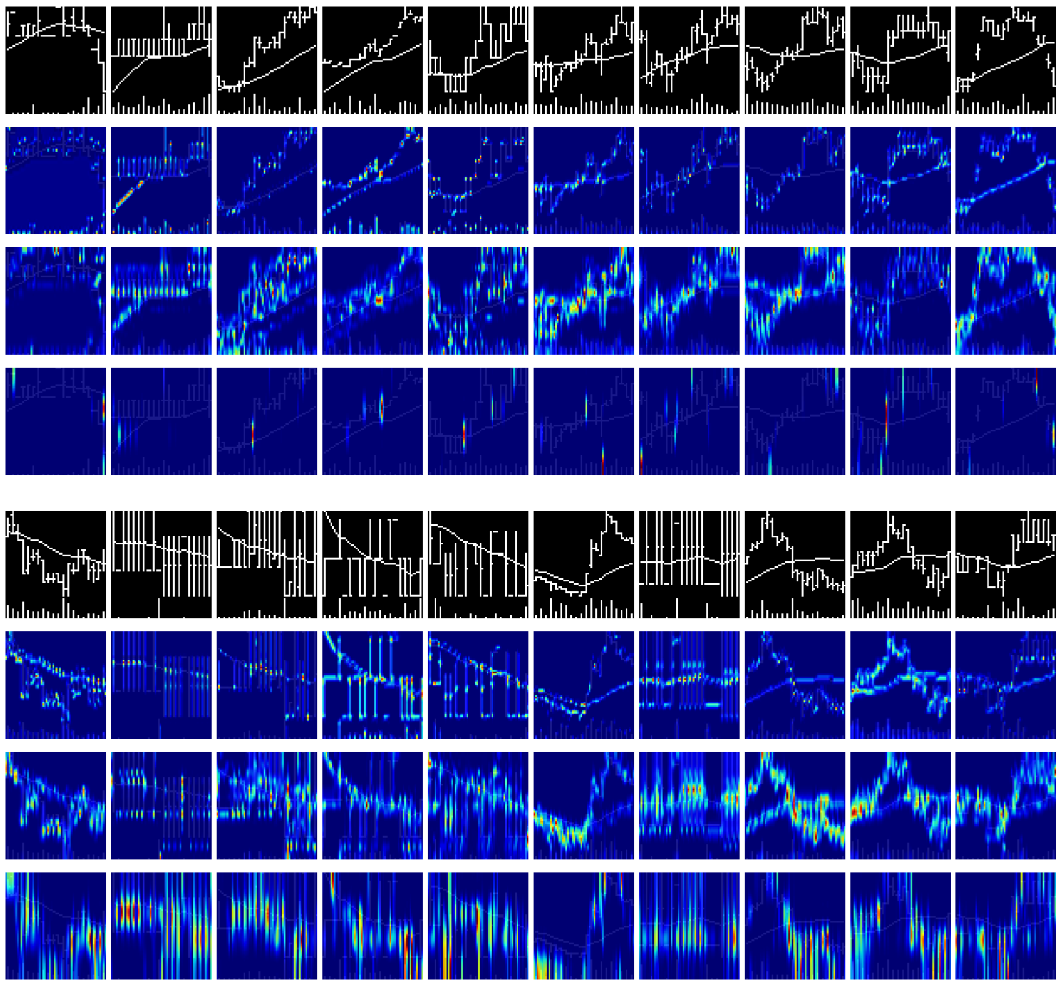


Figure 4: Plots of Grad-CAM for predicting 20-day return horizon: upper half for images classified as "up" trend and lower half for images classified as "down" trend.

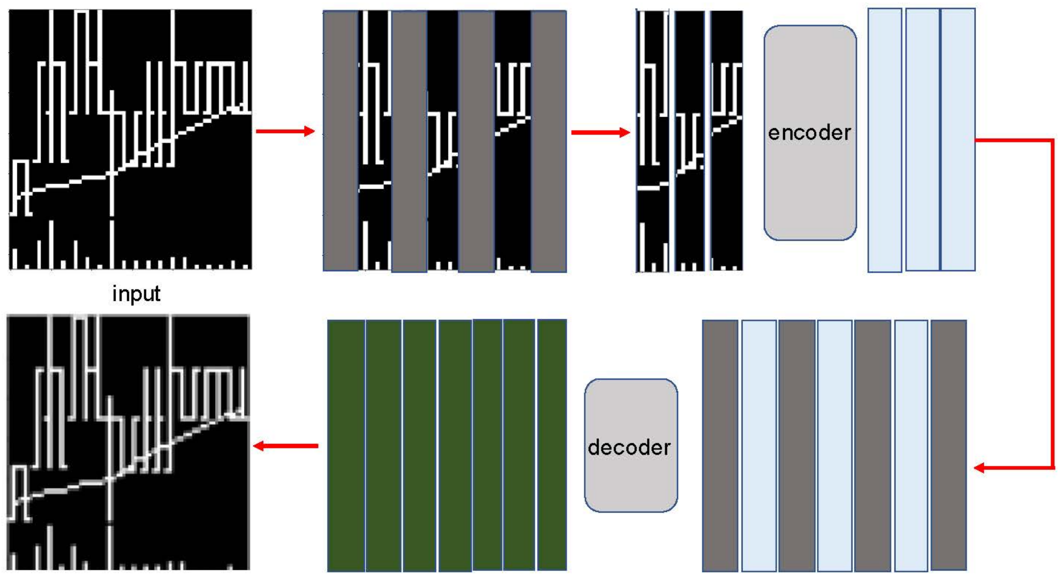


Figure 5: MAE architecture

4 Exploring the Effectiveness of Self-Supervised and Masked Autoencoders Learning when Predicting Imaging Market Data

4.1 The motivation of Self-Supervised and Masked Autoencoders

In this section, we will explore the feasibility of employing self-supervised learning in our task by adapting Masked Autoencoders (MAE) model [8]. Deep leaning models can reach or even surpass human-level performance in many tasks. Due to their highly satisfactory effectiveness, deep models are widely used in both Natural Language Processing and Computer Vision. However, the training of human-outperforming models relies heavily on a large amount of labeled data, which are very expensive to obtain. On the other hand, the large volume of unlabeled data can be continuously generated and easily collected. As a result, relevant works aiming at exploring how to leverage unlabeled data to train well-performing models has constantly emerged in recent years.

The basic motivation of MAE is very simple and straightforward. Given partial signal observations, how to make the model can automatically predict the complete signal corresponding to those observations. Obviously, MAE is naturally applicable in self-supervised learning tasks, which is expected to learn representative features by fitting a large amount of unsupervised data, thus allowing to better provide an desirable pre-trained model for downstream tasks.

Inspired by that, we would like to explore the feasibility of employing self-supervised learning in our task by adapting MAE model. Figure 5 illustrates the pipeline of overall MAE framework designed accordingly for the task. The self-supervised pre-trained task is expected to extract representative features that can effectively guide the learning of downstream tasks from a large volume of unlabeled data.

4.2 The approach of MAE

Masking. To prevent self-supervised models to just capture low-level images statistics from those data, largely reducing redundancy is necessarily required. Since natural images in ImageNet dataset [4] used for the training of original MAE models [8] are mostly generated from the real world, and their content usually consists of an object and its corresponding background, natural images are hence highly redundant both horizontally and vertically, *i.e.*, spatially, as [8] mentioned. Therefore, the generation of random patches in natural images for training MAE models should consider both horizontal and vertical directions. However, it should be noted that the main component of the images used in our task is a concatenation of daily OHLC and volume bars over consecutive 20-day intervals (approximately monthly), where both two bars are almost irrelevant in vertical direction. According to this property, different from the training of original MAE model, we only slice the train images along the horizontal direction to produce patches as shown in Figure 5. In particular, complete cuts are performed vertically, *i.e.*, the height of patches $Patch_H$ is that of the image H, and partial cuts are performed horizontally, where the width of patches $Patch_W$ is significantly smaller than that of the image W. Then, we randomly mask fixed ratio (*i.e.*, mask ratio) patches and get the latent representation after inputting these unmasked patches to the encoder module.

MAE encoder. Same to the setting of [8], the encoder is a ViT model that embeds unmasked patches by a linear projection with added positional embeddings, and then processed the resulting set via a series of Transformer blocks. The benefits of designed encoder are two-fold. The first one is to make it possible to train large encoders with by using only a fraction of compute and memory and the other is to force encoder extract independent and meaningful representations from unmasked patches during training phase. Once the MAE is trained well, the encoder module can be served as a well-trained pretrained model that produces intermediate features to guide the learning of downstream tasks.

MAE decoder. The encoded patches and mask tokens are combined together as the input to the MAE decoder. Similar to the original MAE architecture, each mask token is a learned vector that indicates the presence of a missing patch to be predicted. To achieve that, the location information in the image about mask tokens should be given to recovery the correct order of both encoded patches and mask tokens. Particularly, positional embeddings to all patches and mask tokens are leveraged to preserve the location information. The decoder is also based on Transformer blocks. We need to note that the MAE decoder is only used during pre-training to perform the image reconstruction task. Therefore, unlike traditional Autoencoders, the structure of the decoder does not have to be the same as the encoder architecture. In fact, in our design, the decoder architecture is quiet more simplistic than the encoder. By the decoder, the embeded patches and mask tokens can be mapped to the input image space to get the reconstructed image.

Reconstruction objective. The reconstruction objective is totally same as the setting of common Autoencoders. The MAE reconstructs the input in pixel level, we adopt the mean square error (MSE) of the input and reconstructed image as the training objective to optimize the parameters of the whole MAE model.

$$L_{train} = E_{x \sim \mathrm{X}} \left\| x - f_{MAE}(x) \right\|_2^2 \tag{4.1}$$

4.2.1 Setup

We first introduce the implement details and hyper-parameters for MAE training. Due to difference between the images used in our task and ImageNet, we make some modifications. We use ViT-tiny model as the main block for MAE encoder instead of ViT-base. In addition, we increase the pretraining epoch from 400 to 2000, and the warmup epoch from 40 to 100. The whole training epoch is set as 1000. Then, we increase the batch size for training the classifier from 1024 to 2048 to more efficiently utilize the GPU memory and accelerate training process. Since our images are generated by specific methods and quiet more simple than ImageNet, we do not apply general training strategies for image classification, such as augmentation, label smoothing, cutmix, *etc*., implemented in the original MAE model [8], to train our MAE models. As mentioned before, mask ratio and the width of patches ($Patch_W$) are closely related to the input image and MAE architecture. We chose three values for mask ratio, *i.e.*, 0.75, 0.50, and 0.25. Similarly, for $Patch_W$, we tested three values as well, including 2, 4, and 6.

Once the whole MAE is well-trained by the reconstructed loss, only the encoder is used to produce image representations for classification in our task. Specifically, the output of encoder is followed by a fully connected layer with 2 output neurons for our binary prediction classification task. We believe that a well-trained MAE model is able to capture a good feature representations used for training downstream tasks. Therefore, in order not to cause damage to the potential feature representation mapped by encoder, we choose to freeze the parameters of encoder and only optimize the parameters of the new fully connected layer during the finetuning process. In practice, the whole finetuning epoch is set as 50 and we use Adam optimizer with learning rate 0.001. The value of batch size is 512 and the weight decay is 0.05.

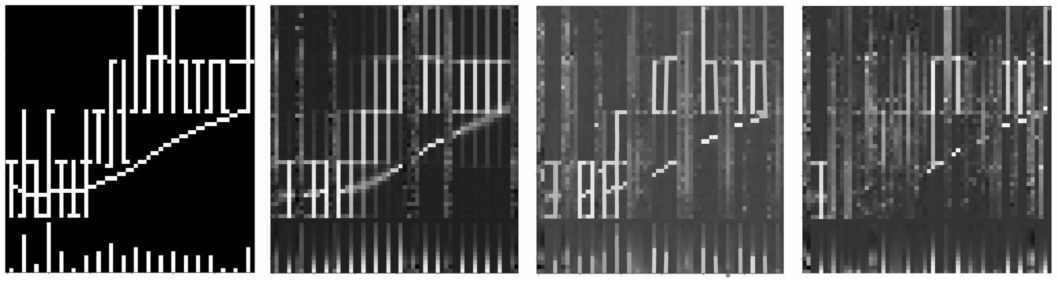


Figure 6: MAE Reconstruction.

4.2.2 Results

4.3 Evaluation for MAE

Reconstruction effect. Figure 6 compares the original image to the reconstruction images generated by the well-trained MAE model with $Patch_W$ equals 2. The image in the first column is the original image, followed in order by reconstructed image generated by MAE model with mask ratio equals 0.75, 0.50, and 0.25. In general, the reconstructed image and the original image are visually similar, but the reconstructed image does not achieve a desirable effect in the generation of some image details, such as the component OHLC bars part, so the reconstructed image looks blurrier from the visual point of view, which shows that the change of image frequency domain is fairly smooth. Moreover, we can also find that the reconstruction effect obtained by using a larger mask ratio(*i.e.*, 0.75), during training is visually similar to that of a smaller mask ratio (*i.e.*, 0.25), which is consistent with the design motivation of MAE models. This is because a larger mask ratio actually encourages the MAE model not to learn statistical information limited to the original image space, but to capture meaningful features that represent the corresponding potential distribution of the image.

Table 3: Accuracy for the classification model finetuned on pretrained MAE encoder.

	Mask Ratio (0.25)			Mask Ratio (0.50)			Mask Ratio (0.75)		
$Patch_W$	2	4	6	2	4	6	2	4	6
5-day	0.503	0.503	**0.506**	0.503	0.498	0.496	0.496	0.502	0.496
20-day	0.492	0.509	0.506	0.495	0.508	0.492	**0.510**	**0.510**	0.509
60-day	0.503	0.500	0.502	**0.506**	0.503	0.504	0.493	0.498	0.498

Classification performance based on the pretrained MAE model. Table 3 shows the classification accuracy on various models finetuned on pretrained MAE encoders with different mask ratio and patch width. In most cases, the classification accuracy of MAE finetuning is slightly higher than the correct rate of completely random guesses (*i.e.*, 0.5), but the obtained result is still very marginal. We believe that it may be due to the fact that such artificially constructed stock trend data based on specific rules are highly correlated in time series, and therefore their use for self-supervised pretraining of MAE is not well suited. In addition, together with Table 1 indicating the classification accuracy on the completely supervised scheme, both the classification performance is not desirable, *i.e.*, slightly higher than 0.5, maybe the discriminable pattern present in the data itself is not obvious enough to train a model that can distinguish well between the two classes.

References

[1] Abien Fred Agarap. Deep learning using rectified linear units (relu). *CoRR*, abs/1803.08375, 2018.

[2] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[3] Antoine d'Acremont, Ronan Fablet, Alexandre Baussard, and Guillaume Quin. Cnn-based target recognition and identification for infrared imaging in defense systems. *Sensors*, 19(9):2040, 2019.

[4] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2009), 20-25 June 2009, Miami, Florida, USA*, pages 248–255. IEEE Computer Society, 2009.

[5] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In Jill Burstein, Christy Doran, and Thamar Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186. Association for Computational Linguistics, 2019.

[6] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021.

[7] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *AISTATS*, volume 9 of *JMLR Proceedings*, pages 249–256. JMLR.org, 2010.

[8] Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross B. Girshick. Masked autoencoders are scalable vision learners. *CoRR*, abs/2111.06377, 2021.

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778. IEEE Computer Society, 2016.

[10] Jingwen Jiang, Bryan T. Kelly, and Dacheng Xiu. (re-)imag(in)ing price trends. *Chicago Booth Research Paper*, No. 21-01, December 1, 2020.

[11] Ramprasaath R. Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *ICCV*, pages 618–626. IEEE Computer Society, 2017.

[12] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015.

[13] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5998–6008, 2017.

[14] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In William W. Cohen, Andrew McCallum, and Sam T. Roweis, editors, *Machine Learning, Proceedings of the Twenty-Fifth International Conference (ICML 2008), Helsinki, Finland, June 5-9, 2008*, volume 307 of *ACM International Conference Proceeding Series*, pages 1096–1103. ACM, 2008.