CLASSIFYING STOCK RETURNS USING HIGH-FREQUENCY FUNDAMENTAL FACTORS AND CONVOLUTIONAL NEURAL NETWORKS

A study on five U.S. equities between 2013 and 2017

AXEL KOTNIK

DENIS DVINSKIKH

MFIN Thesis

Stockholm School of Economics

2021

CLASSIFYING STOCK RETURNS USING HIGH-FREQUNCY FUNDAMENTAL FACTORS AND CONVOLUTIONAL NEURAL NETWORKS

Abstract:

We evaluate the usefulness of high-frequency fundamental factor exposures of five US equities, between 2013 and 2017, as features for classifying and predicting the binary movements of the same stocks in 5-minute and 20-day intervals using Convolutional Neural Networks (CNN). After plotting rolling factor betas (Market, HML, SMB) and the close price of a given stock in the corresponding intervals, these time series are converted into images as Gramian Angular Difference Fields (GADF) and then concatenated to be fed to the CNN as input. Two types of convolutional neural networks are trained on these images and used for a binary classification task of determining whether the close price is likely to increase or decrease in the consecutive time unit. For comparison, the same analysis is conducted with technical indicators (RSI, EMA, %K) and a combination of the two (Market, HML, RSI). The results of this paper show moderate performance of the trained CNN, achieving a maximum accuracy on test data of 54.7% for a 20-day interval using images with a combination of both technical indicators and fundamental factors. For further research, we suggest using a longer forecast and classification horizon, and exploring alternate ways to linear regression for high-frequency beta estimation for fundamental factors.

Keywords:

Convolution neural networks, technical indicators, fundamental factors, high-frequency stock prices, classification, prediction

Authors:

Axel Kotnik (24179)

Denis Dvinskikh (41924)

Supervisor:

Riccardo Sabbatucci, Assistant Professor, Department of Finance

Examiner:

Dong Yan, Assistant Professor, Department of Finance

Master Thesis
Master Program in Finance
Stockholm School of Economics

Table of Contents

1. Introduction

The potential gains of successful predictions of the stock market have led people to attempt to forecast stock prices for many decades. The commonly held view since the early 1960s, via the capital asset pricing model (Sharpe, 1964) and efficient market hypothesis (Fama, 1970), is that returns are proportional to the asset's exposure to general market risk and that all publicly available information about an asset is incorporated into its price immediately upon release. The consequence is that arbitrage, i.e., risk-free returns, cannot be achieved and that one cannot beat the market in the long term. The capital asset pricing model has been extended by Fama and French (Fama & French, 1993) to include two other factors that drive returns beside the market in aggregate, namely having a high book-to-market ratio and being a smaller sized firm. Since then, many factors have been proposed, tested and rejected in the "factor zoo" (Feng, Giglio, & Xiu, 2020).

In the short term, asset pricing theory is less useful as price movements are characterized by high dimensionality, e.g., in the form of idiosyncratic events, large orders, rumors or irrational trading. However, the emergence of statistical analysis and machine learning tools has helped researchers in discerning useful information from noise and has instigated the scientific community to explore many new tools to understand and capture returns in the short term (Kara, Boyacioglu, & Baykan, 2011). Deep learning and neural networks have during the last 10 years played an integral part in this pursuit (Hu, Zhao, & Khushi, 2021) (Sezer, Gudelek, & Ozbayoglu, 2020). Our intention with this paper is to shed light on the intersection between long term and short-term stock price predictions. With the help of high-frequency versions of Fama and French's three factors, normally used to predict asset prices in the long term, and the promising feature extraction capabilities of convolutional neural networks (CNN), shown to effectively predict asset returns in the short term, we hope to uncover the respective contributions of statistical analysis versus fundamental analysis to the predictability of one-directional movements of short-term stock prices.

2. Literature Review

2.1 Deep learning in asset pricing

Research aiming to predict asset prices using deep learning has risen exponentially since 2015, as mapped out by Hu et al. (Hu, Zhao, & Khushi, 2021) and Sezer et al. (Sezer, Gudelek, & Ozbayoglu, 2020). This likely stems from the increased attention these methods have gained following their success in other fields of science alongside the democratization of deep learning algorithms from tools like Keras. Most commonly, the exercise is to forecast a single financial time series from its own history to predict the subsequent rise or drop in value, and thereafter construct profitable trading strategies with successful such algorithms. The majority of deep learning models evaluated by Sezer et al. outperformed machine learning models.

Due to the volatility and non-linearity of stock prices in the short term, deep learning is well-suited to handle this issue and most researchers use high-frequency data windows

of hours or days to make consecutive predictions. Sezer et al. (Sezer, Gudelek, & Ozbayoglu, 2020), whose paper mapped 140 deep learning publications in financial time series prediction from 2005 to 2019, showed that memory-based recurring neural networks (RNN) was the most popular model and was used in 62% of papers; a natural choice because of the ordinal nature of time series. CNN follows and was used in 21% of the papers, of which a total of 11 attempted to predict stock price movements. Other papers quantify financial news sentiment and analyze this along historical stock prices with deep learning for movement predictions (Ding, Zhang, Liu, & Duan, 2015) (Cai, Feng, Deng, Ming, & Shan, 2018) (Kraus & Feuerriegel, 2017) (Maqsood, o.a., 2020).

Most research using CNN on images of time series has been done in the last three years and has shown great results. In ten of the papers reviewed by Hu et al. (Hu, Zhao, & Khushi, 2021), the accuracy of CNN-based stock prediction models averages ~70% and range from 55% to 95% – significantly higher than guessing. Correspondingly, RNN achieved a ~68% average accuracy, LTSM ~67%, and DNN ~68%. Di Persio & Honchar find their CNN to outperform MLP and RNN by 1.5% accuracy (Di Persio & Honchar, 2016). Gunduz et al. find their CNN to consistently predict intraday stock movements on the Istanbul stock exchange with 55% accuracy. Without accounting for trading costs and other constraints, any model whose predictions are correct more than 50% of times can be considered profitable.

In the majority of CNN, technical indicators have been used as explanatory variables for stock price forecasting (Gunduz, Yaslan, & Cataltepe, 2017) (Liu, Zeng, Yang, & Carrio, 27–28 August 2018) (Gudelek, Ozbayoglu, & Boluk, 2017) (Ozbayoglu & Sezer, 2018) (Sim, Kim, & Ahn, 2019) and did, in their respective simulations, outperform the common buy-and-hold strategies. Technical factors are often constructed in the short term and include moving averages, RSI, Williams %R, etc. However, technical factors do not necessarily provide additional explanatory value. Sim et al. (Sim, Kim, & Ahn, 2019) found that their best model was a CNN with only the close price as input (accuracy ~65%), as opposed to models with up to nine technical indicators (accuracy <60%). They hypothesize that this is due to many technical indicators being similar in appearance to the stock price, thus adding no spatial information for the CNN to process.

Besides handling spatial information especially well, CNN is also a time invariant neural network and does not account for the order of information, like e.g. RNN. Due to the noise and little correlation between short term returns, CNN would arguably lose less predictive power when used for short term predictions. As a consequence, it is common to use daily observations or even minutes like Selvin et al. (Selvin, Ravi, Gopalakrishnan, & Menon, 2017) or Sim et al. (Sim, Kim, & Ahn, 2019).

A less common approach used by some (Hoseinzade & Haratizadeh, 2019) (Enke & Zhong, 2016) is to include economic variables as inputs, including world indices, foreign exchange rates, commodities, futures and data from big companies. Hoseinzade & Haratizadeh (Hoseinzade & Haratizadeh, 2019) use both 2D and 3D input tensors; the last dimension being different markets to

account for individual traits of each stock index. They include a total of 82 economic and technical variables in their CNN input. They benchmarked the CNN performance against other literature algorithms: an ANN being fed features extracted by PCA in Enke & Zhong (Enke & Zhong, 2016), a shallow ANN classifying on technical indicators (Kara, Boyacioglu, & Baykan, 2011), and a CNN being fed technical indicators only (Gunduz, Yaslan, & Cataltepe, 2017). Their own algorithms averaged an F-measure of ~0.5 over several world indices compared to ~0.42, ~0.42 and ~0.39 for their benchmark models. The authors conclude that adding more variables did not improve predictability (in the case of the 3D model, separating models per market also worsened performance), but that the depth of their network was the cause for outperforming others.

Another aspect that needs consideration is the presentation of inputs to neural networks for optimal performance. Gramian Angular Fields (GAF) is a method of encoding time series into images proposed by Wang & Oates in 2015 (Wang & Oates, 2015), developed as a means for computer vision to classify time series more efficiently after its success with image recognition. Feeding the time series as GAF to the CNN provides several advantages (Chen & Tsai, 2020), including: (1) preserving temporal dependency as time increases when moving from top-left to bottom-right; (2) it contains intertemporal correlations; (3) the original data is stored in the primary diagonal of the picture and one can theoretically reconstruct the time series from high-level features learned by the deep network. Chen & Tsai (Chen & Tsai, 2020) also suggest using candlesticks as input images for GAF, which is then fed to the CNN, as this yields greater spatial variety to the images and improved their model's performance. Sezer & Ozbayoglu (2020) emphasize the weight CNN places on local features and adjacent pixels, and therefore suggest that one should choose neighboring data points and presentation carefully.

An alternate, useful way to present explanatory variables to CNN is using dummies variables and labelling them as "1" if their values pass some threshold, and "0" otherwise. Yang et al. obtained better results from their CNN when presenting continuous technical indicators as binary trend signals (Yang, Zhai, & Tao, 2020). Unlike this approach, which exhibits the values and intertemporal correlations of a single variable during some lags, some research papers (Ozbayoglu & Sezer, 2018) (Hoseinzade & Haratizadeh, 2019) construct matrices where one dimension represents lagged time values and the other represents the values of different indicators each day. It is our idea, inspired by Sim et al. (Sim, Kim, & Ahn, 2019), that the spatial uniqueness of linear, continuous graphs can contribute to CNN performance and we choose to follow this method.

2.2 Deep learning with fundamental factors

The literature is sparse in this specific field of research. In the review of Hu et al., one paper used neural networks and fundamental factors to predict asset price movements. Abe & Nakamaya (Abe & Nakamaya, 2018) use deep learning and accounting ratios to predict the one-month-ahead stock returns in the

cross-section of the Japanese stock market from 2002 to 2016. They extract fundamental information about stocks from their quarterly reports (e.g., book-to-market ratio, ROIC, current ratio, Sales-to-price, etc.), and update their factors once every month. Their models are fully connected feed-forward deep neural networks benchmarked against shallow networks and found that deep versions outperform the shallow ones marginally.

Zhou (Zhou, 2019) develops a novel two-layered LSTM recurrent neural network and MLP model combination to predict the next day's return from 80 days of past returns, along 15 annual accounting figures (ROE, investment-to-capital, etc.), for 99% of listed US firms between 1981 and 2017. Seventeen stock portfolios are constructed based on fundamental factors whose 80 days' past returns are being fed to a CNN to create a trading strategy that goes long (short) the three portfolios with highest probability of having a positive (negative) return on the 81st day. Realized annualized returns before trading costs of 34.33% are obtained during this time period. Zhou also notes that returns diminish after 2010, supposedly because of the democratization of deep learning algorithm trading that have traded away such arbitrage opportunities. To our knowledge, no one has modeled individual stock returns from their exposure to fundamental factors, in the short term, with deep learning.

Deep learning has also been used for improved fundamental factor construction. Feng et al. (Feng, Polson, & Xu, Deep Learning in Characteristics-Sorted Factor Models, 2019) consider factor models as deep learning architectures, in the way that (1) firm characteristics are inputs, (2) risk factors are hidden layers, and (3) excess returns are outputs. Using the improved and non-linear Fama French factors that their hidden layers represent, and kernel weights that represent beta exposures, they fit the cross-section of stock returns better than the original works.

2.3 Hyperparameter and network optimization

Neural networks contain several hyperparameters that need to be tuned for efficient learning and optimal performance. As hyperparameters cannot be learned during the training process, they need to be correctly specified in advance. Hyperparameters dictate the model's complexity, speed of convergence, learning rate, capacity of the model and the training specifications in terms of batch size, number of epochs and certain types of activation functions.

How has hyperparameter tuning been approached historically? Considering the atypical format of stock price pictures versus natural objects, hyperparameter tuning should be closely assessed. Some, including Di Persio & Honchar (2016), have used a sequential model-based optimization (SMBO) approach to tuning. In their case, they use a tree-structured Parzen Estimator.

As a starting point, smaller kernel sizes are preferred over larger ones to capture more detailed, local information of images and feature maps. Secondly, kernels with odd dimensions are preferred as all the previous layer pixels are symmetrically positioned around the new centered pixel which alleviates the modeler of accounting for spatial distortions.

Many papers (Hoseinzade & Haratizadeh, 2019) (Sim, Kim, & Ahn, 2019) (Chen & Tsai, 2020) (Di Persio & Honchar, 2016) (Liu, Zhang, & Ma, 2017) (Sim, Kim, & Ahn, 2019) constructing CNN for financial predictions base their models on the classic LeNet-5 architecture, which consists of (1) a convolutional layer followed by a pooling layer, (2) another similar convolutional layered followed by a pooling layer, (3) a flattening layer, and (4) a fully connected output layer (LeCun, et al., 1989). When it comes to our data set, a type of financial time series, Hoseinzade & Haratizadeh (Hoseinzade & Haratizadeh, 2019) argue that the 3 x 3 or 5 x 5 filters that are industry standard in image processing may not necessarily be the optimal choice. They consider the idea of candlesticks, which serve to combine several traits of a stock price at a moment in time into a single, higher level feature. Thus, a 1 x 82 kernel is constructed that strides across 82 input variables for each day. Their complete setup is as follows: a convolutional layer of eight 1 x 82 filters, after which there are two convolutional layers with eight 3 × 1 filters, each followed by a layer of 2 × 1 max-pooling, lastly followed by a flattening operation that is fed into a fully connected layer. A similar configuration is used by Yang et al. (Yang, Zhai, & Tao, 2020) and Gunduz et al. (Gunduz, Yaslan, & Cataltepe, 2017), but where two convolutional operations are done on the input image in parallel that are the concatenated.

Several papers (Yang, Zhai, & Tao, 2020) (Yang, Zhu, & Huang, 2018) (Chen & Tsai, 2020) recommend not using any pooling operations as it mostly incurs information loss on financial markets data given their suspicion that such series may be truncated. This applies to dropout rates as well, an overfitting mitigant that should be set to zero in financial time series for best performance according to Sim et al. (Sim, Kim, & Ahn, 2019).

The choice of optimizer is not uniform – Adam is used by many (Liu, Zhang, & Ma, 2017), AdaDelta by others (Gunduz, Yaslan, & Cataltepe, 2017) (Gudelek, Ozbayoglu, & Boluk, 2017) (Di Persio & Honchar, 2016), and SGD is considered superior by some. Activation functions also vary from paper to paper, but most seem to prefer ReLU for hidden layers (Sim, Kim, & Ahn, 2019) and tanh (Liu, Zhang, & Ma, 2017), softmax (Sezer & Ozbayoglu, 2018) (Yang, Zhai, & Tao, 2020) or sigmoid for output. The intuitive basis for sigmoid or softmax activation is that it takes a value on the continuous scale from 0 to 1, which in our case can be interpreted as the probability of the stock to move upwards. Softmax can be considered even more intuitive as its outputs are mutually exclusive and sum up to one over the different classes.

Chen & Tsai (Chen & Tsai, 2020) developed a CNN architecture especially tailored to interpret GAF as input. Their input and target variable for prediction is the EUR/USD foreign exchange rate in candlestick patterns from 2010 to 2017, and their experimental results achieve 90.7% accuracy. They use Adam optimizer, batch size 64 and 300 epochs. They observe that the simple LeNet architecture works well with the GAF-CNN and therefore mimic that

design; two convolutional layers with 16 kernels and one fully connected layer with 128 nodes.

2.4 Contribution

Fundamental factors, which consider the fundamental financial performance of companies as drivers of returns, have been found useful for predicting cross-sectional returns in the long term. Technical factors have instead been relied upon for the prediction of individual stock returns in the short term. At the same time, CNN have not successfully harnessed technical factors as inputs for stock price predictions, speculatively due to their similarity in appearance to stock prices themselves; as discussed, CNN are good at processing spatial information. We take a new approach to address the shortcomings of both aforementioned topics. By continuously measuring a given stock's exposure to high-frequency estimations of the three Fama French factors, as constructed by Aït-Sahalia et al. (Aït-Sahalia, Kalnina, & Xiu, 2020), we generate fundamental features that are inherently linked to the stock price itself in the short term while also providing input that behaves dissimilar to the stock price, unlike technical factors. In this way, we hope to provide the spatial diversity that CNN are specialized to extract features from while uncovering the potential link between short term returns and fundamental factors. To benchmark the independent explanatory value of fundamental factors, we also train identical models on technical factors for comparison. To measure the joint informativeness of fundamental and technical factors, we also benchmark against a data set with a combination of fundamental and technical factors.

3. Theoretical background

3.1 Machine learning and neural networks

Within artificial intelligence, machine learning is the concept of a computer learning to perform a specific task without being explicitly programmed to. This can be done in a *supervised* fashion, where algorithms are given already labeled data find patterns in (linear regression, logistic regression, support vector machines, etc.); in an *unsupervised* fashion, when the program finds structures in unlabeled data (K-means clustering, principal components analysis, etc.); or in *reinforcement learning*, where the machine learns from feedback in real and synthetic environments. Neural networks are a subset of algorithms within the machine learning space that stem from neurophysiological research in the mid-20th century when scientists began to understand the connectivity mechanisms inherent to neurons in biology and subsequently model them on computers. Neural networks pass information between nodes ("neurons") and subsequently learn the importance of each connection for a specific prediction and adjust the value of these weights ("neural connections") accordingly. The advantage of neural networks lies in their flexibility and ability to detect non-linear relationships in data.

3.2 Convolutional Neural Networks

A Convolutional Neural Network (CNN) is one type of neural network that was developed for computer vision, inspired by the structure of the visual cortex of cats

(Lindsay, 2020). The first convolution of a CNN recognizes local edges and contours of raw input image, which is then passed on to sequential layers that combine these into features such as “ears” or “wheels”. The final layer classifies the image based on these features in aggregate, making predictions like “animal” or “car”. The prototype model was developed by Fukushima in 1980 (Fukushima, 1980) who was inspired by the aforementioned biological findings in the 1950s. One of the most prominent CNN, and the first to be used commercially when banks deployed it to recognize hand-written letters, was developed by LeCun et al. in 1998 (LeCun, Bottou, Bengio, & Haffner, 1998). Today, a variety of CNN configurations are available and applied.

Components of a CNN

CNN architectures come in many different variants, and the literature presents a variety of options for exploration. One of the most famous CNN architectures is the LeNet-5, which consists of convolutional layers, pooling layers and fully connected layers. Each convolutional layer picks up essential features of images, such as edges and shapes through kernels, which are then passed to pooling layers, that simplify and reduce the dimensions of the supplied images for quicker optimization of the neural network. The depth of convolutional layers varies greatly, but a common approach is to use 2 to 5 convolutional layers. Following convolutional and pooling layers, fully connected layers and an output layer comes last to final generate the final prediction as global semantic information (Gu, et al., 2018).

Convolutional layer

A convolutional layer in a neural network performs a convolutional operation on the input matrix. It is done by passing a filter to an input that results in an activation. In practical terms, the filter slides across the entire input image and passes on higher level information as specified by the activation function onto a feature map, which acts as input image for the next layer in the model. Each filter uses a shared set of weights that are optimized and updated during training of the neural network. A CNN thus learns the optimal weights of its filters given a specific data set. If input I is a $N \times N$ matrix, and a convolutional filter of size $F \times F$ (where $N > F$) is applied on each entry of the matrix, a corresponding weight w is generated. Then the output of the convolutional operation is calculated through equation (1):

$$v_{i,j}^{I+1} = \delta\left(\sum_{k=0}^{F-1}\sum_{m=0}^{F-1} w_{k,m}\, v_{i+k,j+m}^{I}\right) \qquad (1)$$

In equation (1), δ represents an activation function, $v_{i,j}^{I+1}$ is the value at the i^{th} row and j^{th} column in the resulting output matrix I+1, and $w_{k,m}$ is the weight assigned to the k^{th} row and m^{th} column of the filter.

Activation functions

The activation function in a hidden layer is an essential part of neural networks. For hidden layers there are typically three types of activation functions used: Rectified Linear Activation (ReLU), sigmoid (σ) and Hyperbolic Tangent (tanh), whilst for output

layers, softmax (f) or sigmoid (σ) is most frequently employed:

$$ReLU(x) = \max(0, x) \quad (2)$$

$$\sigma(x) = \frac{1}{1+e^{-x}} \quad (3)$$

$$tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}} \quad (4)$$

$$f(x) = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}} \quad (5)$$

The most common use of activation functions in convolutional neural networks is via the ReLU activation function to avoid issues with the vanishing gradient problem. Typically, the same activation function is used in all hidden layers. In the output layer for categorical classification, a softmax or sigmoid activation function is used.

Batch normalization layer

Batch normalization is a technique implemented in neural networks that standardizes the input that is passed onto the next layer. The standardization is important because during training the model is updated backwards, and since layers take input from previous layers, the inputs might alternate for each batch. The standardization is done by rescaling the input data to have a mean of zero and a standard deviation of 1. This results in subsequent layers not having to make assumptions about the distribution of the inputs when updating the weights. Batch normalization significantly reduces the number of epochs required to train the data, resulting in better performance.

Pooling layer

As overfitting is a reoccurring issue when training neural networks, pooling layers are used in convolutional neural networks to reduce the spatial size of the features and significantly reduce the numbers of parameters that need training. Essentially, a pooling layer reduces the dimensions of the feature maps. A pooling layer slides a filter of selected size, most commonly 2×2, across the input image and performs an operation on the selected values, most commonly max pooling or average pooling. Max pooling operation takes the maximum values from the input from the pooling filter input and average pooling takes the average value. Thus, given a $N \times N$ matrix that is passed to a pooling layer with a $k \times k$ dimensional pooling filter is applied, the resulting matrix will have dimensions of $\frac{N}{k} \times \frac{N}{k}$.

Flattening layer

A flattening layer is placed between the last convolutional layers and the fully connected layers, in order transform the matrix into an array with a single column.

Fully connected layer

Fully connected layers perform two operations on the input that is passed to them. Firstly, a fully connected layer performs a linear transformation in the form of a dot product of input matrix values and weights, and secondly, applies a non-linear transformation:

$$v_i^{j+1} = \sigma\left(\sum_k v_k^j \, w_{k,i}^j\right) \quad (6)$$

In equation (6), v_i^{j+1} represents the value of the i^{th} neuron at the $j+1^{st}$ layer, and $w_{k,i}^{j}$ is a weight between the connections of the k^{th} neuron from the j^{th} layer and the i^{th} neuron from the $j+1^{st}$ layer.

Dropout layer

Another remedy for overfitting in neural networks is the application of dropout layers. Small datasets or an abundance of layers will cause the neural network to learn statistical noise in the data and not generalize well to new, unseen data. One alternative is to train an ensemble of networks and average out the results, but this is computationally expensive and a dropout layer can instead reduce the complexity of the model by randomly selecting nodes to omit. Dropout layers are implemented on a per-layer basis. A hyper-parameter with the probability of retaining a node must be specified.

3.3 Gramian Angular Fields

Gramian Angular Field (GAF) images are RGB-channel representations of univariate data. A time series has first been converted into polar coordinates, whose angles by various operations have then been converted into a symmetry matrix, yielding a GAF.

GAFs are constructed using a two-step approach. Initially, given a time series $T = \{t_1, t_2, t_3, \dots, t_n\}$, the time series T is rescaled using min-max scaling:

$$t_{i,sc} = \frac{\big(t_i - max(T)\big) + \big(t_i - min(T)\big)}{max(T) - min(T)} \quad (7)$$

Given that the rescaled values of t_{sc}^{i} lie between -1 and 1, we can apply the arccos operator resulting in values that are between 0 and π. The second component required for converting rescaled values to polar coordinates are the time stamps which are obtained by dividing the time stamp i by the total number of observations n, and saved as the radius for the polar representation:

$$\begin{cases} \varphi_i = arccos\big(t_{i,sc}\big) \\ r_i = i/n \end{cases} \quad (8)$$

The GAF can take on the form of a difference operator or a summation operator denoted by Gramian Angular Difference Field (GADF) or Gramian Angular Summation Field (GASF) respectively, where the difference lies in the use of a cosinus operator when constructing the Gram matrix for the GASF and the sinus operator for the GADF. In this paper, GADF are used.

Once the angles and radii are obtained, the second step of the time series to GAF transformation can be done:

$$GADF = \sqrt{I - T_s c^{2\prime}} \cdot T_s c - T_s c' \cdot \sqrt{I - T_s c^2} \quad (9)$$

$$GADF = \begin{bmatrix} sin(\varphi_1 + \varphi_1) & \cdots & sin(\varphi_1 + \varphi_n) \\ \vdots & \ddots & \vdots \\ sin(\varphi_n + \varphi_1) & \cdots & sin(\varphi_n + \varphi_n) \end{bmatrix} \quad (10)$$

It is important to note that number of channels of the input image must match the

depth of the convolutional kernel dimensions. If one uses GAF on the red-green-blue (RBG) scale, the depth of the kernel matrix must be three; otherwise, it cannot capture the information between the color scales.

3.4 Technical indicators

Technical analysis of stock prices has been around for several decades, and first use of technical analysis for investing dates back to the late 18th century. Much research has monitored the performance of trades following the state of various technical indicators to show whether or not they carry predictive performance, or if they carry information on the momentum of supply and demand pressures on stocks. Brock, Lakonishok & LeBaron (1992) asserted that trading strategies based on moving average and trading-range breaks can be utilized to trade on future returns based on historical data. Hsu & Halgamuge (2007) used technical indicators and news flow to predict increases and decreases in stock prices and achieved 70% accuracy on their dataset. Shynkevich et al. (2017) use 10 technical indicators to evaluate if they contain information on the movement of future stock prices based on historical data. Their findings include that technical indicators indeed carry information on future stock prices and can be used to make predictions. Following the convention of several papers, and for benchmarking against our fundamental factor models, technical indicators will be evaluated as explanatory variables for binary stock price movements (Gunduz, Yaslan, & Cataltepe, 2017) (Liu, Zeng, Yang, & Carrio, 27–28 August 2018) (Gudelek, Ozbayoglu, & Boluk, 2017) (Ozbayoglu & Sezer, 2018) (Sim, Kim, & Ahn, 2019). The following technical indicators have been used for experimentation in this study; however, not all were used as input features for the final models.

Simple Moving Average

The simple moving average (SMA) is an unweighted mean of the previous k data points. Suppose we have observed a time series $t_1, \dots, t_n$ for close prices of a stock. The mean of the previous k data points is calculated through equation (11) and is labeled SMA_k

$$SMA_k = \frac{t_{n-k+1} + t_{n-k+2} + \cdots + t_n}{k} \quad (11)$$

Exponential Moving Average

To address the issue of equal weighting, the exponential moving average (EMA) is employed by assigning a higher weight to recent observations. The weight for older observations decreases exponentially, but never reaches zero. The exponential moving average is calculated recursively according to the following formula for a given time series $t_1, \cdots, t_n$, where α represents a constant smoothing factor and $0 < \alpha < 1$:

$$EMA_t = \begin{cases} t_1, & t = 1 \\ \alpha \times t_n + (1-\alpha) \times EMA_{t-1}, & t > 1 \end{cases} \quad (12)$$

A common choice for the smoothing coefficient α is $2k+1$, where k is the selected window length (usually around 12 to

26 periods) which is used as input for the moving average convergence divergence indicator.

Moving Average Convergence Divergence

The moving average convergence divergence (MACD) is another technical indicator used in stock analysis looking for buy and sell signals. MACD uses long and short window EMA to find changes in an asset's momentum and trends.

$$MACD_n = EMA_n(12) - EMA_n(26) \quad (13)$$

The MACD is used together with a 9-length window EMA to find crossovers that are indicative of a buy or sell signal.

Stochastic Oscillator

The stochastic Indicator (%D) is another indicator that is used to gauge momentum and trends in stock trading. The stochastic indicator is calculated by first finding the ”slow” indicator denoted by $\%K$:

$$\%K = \left(\frac{C - L14}{H14 - L14}\right) \times 100 \quad (14)$$

This is followed by taking the three-period moving average denoted by %D. The stochastic indicator is bound between 0 and 100.

Relative Strength Index

Relative strength index (RSI) is a measure of momentum that evaluates the magnitude of swings in stock prices to identify oversold or underbought stocks. RSI is typically used on a 14-period basis and has a scale between 0 and 100.

$$RSI = 100 - \frac{100}{1 + \frac{\text{average gain}}{\text{average loss}}} \quad (15)$$

The average gain is calculated in two steps. The first average gain and the first average loss is calculated as the sum of gains and losses over the past 14 periods divided by 14. Then the average gain and loss is calculated by:

$$\text{Average gain} = \frac{[\text{previous av. gain} \times 13 + \text{current gain}]}{14} \quad (16)$$

$$\text{Average loss} = \frac{[\text{previous av. loss} \times 13 + \text{current loss}]}{14} \quad (17)$$

3.5 Fundamental factors

Fundamental, systemic risk exposures of assets were first identified by William F. Sharpe, Jack Treynor, John Lintner and Jan Mossin, building on previous work of Harry Markowitz, during the development of the Capital Asset Pricing Model (CAPM). Stock returns were found to be closely related to their respective exposures to market risk. Subsequently, Eugene Fama and Kenneth French (Fama & French, Common risk factors in the returns on stocks and bonds, 1993) identified additional risk components in a given asset’s correlations with portfolios compromised of long positions in small-cap companies and short positions in large-cap companies, as well as portfolios with long

positions in high book-to-market value ratios and short companies with low book-to-market value ratios. Subsequent research by Fama & French (2014) have identified additional risk factors, and with the work of Mark M. Carhart (1997), also the identification of the momentum factor. An overview of fundamental factors and their construction is provided below.

Market

The Market portfolio consist of the value-weighted excess returns of all publicly traded stocks on the NYSE, AMEX and NASDAQ stock exchanges.

SMB

The Small-Minus-Big (SMB) portfolio is constructed by taking the equally weighted average returns of the top three deciles of small stocks minus the average returns of the three top deciles of large-cap stocks.

$$\text{SMB} = \frac{1}{3}(\text{Small Value} + \text{Small Nuetral} + \text{Small Growth}) - \frac{1}{3}(\text{Big Value} + \text{Big Neutral} + \text{Big Growth}) \quad (18)$$

HML

The High-Minus-Low (HML) portfolio is the equally weighted average returns for the top two deciles of stocks with high book-to-market ratios minus the average returns of the top two deciles of stocks with small book-to-market ratios.

$$\text{HML} = \frac{1}{2}(\text{Small Value} + \text{Big Value}) - \frac{1}{2}(\text{Small Growth} + \text{Big Growth}) \quad (19)$$

CMA

Conservative-Minus-Aggressive (CMA) is the average return of two portfolios with conservative investments less the average return of two portfolios with aggressive investment policies.

$$\text{CMA} = \frac{1}{2}(\text{Small Conservative} + \text{Big Conservative}) - \frac{1}{2}(\text{Small Aggressive} + \text{Big Aggressive}) \quad (20)$$

RMW

Robust-Minus-Weak (RMW) factor is constructed by taking the average returns of two portfolios with robust operating profitability less the average returns of two portfolios with weak operating profitability

$$\text{RMW} = \frac{1}{2}(\text{Small Robust} + \text{Big Robust}) - \frac{1}{2}(\text{Small Weak} + \text{Big Weak}) \quad (21)$$

MOM

The momentum factor specifies that holding stocks that have performed well in recent time periods will likely outperform the returns of last year's worst performing stocks. Momentum factor is the average of the two high prior return portfolios less the average return on two portfolios with low prior returns.

$$\text{MOM} = \frac{1}{2}(\text{Small High} + \text{Big High}) - \frac{1}{2}(\text{Small Low} + \text{Big Low}) \quad (22)$$

4. Methodology

4.1 Data

For this paper, five publicly listed US equities from various industries were selected. These include ExxonMobil (ticker: XOM), one of the largest publicly traded oil and gas companies; Procter & Gamble (ticker: PG), international consumer goods; Netflix (ticker: NFLX), international media and technology company; J.P. Morgan (ticker: JPM), one of the largest US banks; and AT&T (ticker: T), the world's largest telecommunications company. For the equities, 5-minute interval Open, High, Low, Close (OHLC) were obtained and their respective 5-minute returns calculated during the period January 1st 2013 to December 31st 2017. Each stock yielded 98,202 observations over the entire timespan which resulted in a total of 491,010 observations. Adjusted for non-trading days and public holidays pre-trading activity, the data set consisted of 1,259 trading days, with each having 78 five-minute OHLC value for each stock.

To estimate the high-frequency betas of stocks with the fundamental factors Market, SMB, HML, CMA, RMW and MOM, 5-minute high-frequency factor returns were obtained from the public website of Dacheng Xiu, one of the authors behind the estimation of high-frequency factors in the paper of Aït-Sahalia et al. (Aït-Sahalia, Kalnina, & Xiu, 2020). The dataset contains factor portfolio returns at the 5-minute interval for all traded stocks on the NYSE, AMEX and NASDAQ stock exchanges.

4.2 Factor beta construction

The five US stocks' exposures to fundamental factors are estimated using a rolling window regression. Given a sample size T, a window of length m is selected. Once selected, the dataset is divided into $N = T - m + 1$ sub-samples, and for each sub-sample, a regression is performed on the past m observations, thus yielding beta coefficients for the m^{th} datapoint. The window is then rolled onto the $m + 1^{st}$ observation and the returns of the equities are regressed on the 2^{nd} to $m + 1^{st}$ values of the fundamental factors. This process is repeated until the regressions reach the T^{th} value. Given a sample size of 98,202 datapoints for each US equity, the regressions result in 98,202 minus m number of regression outputs. We select m = 100 for the 5-minute intervals, resulting in 98,102 regressions per stock. The regression is specified in equation (23).

$$r_i = \alpha_i + \beta_{1,i} Market + \beta_{2,i} SMB + \beta_{3,i} HML + \beta_{4,i} CMA + \beta_{5,i} RMW + \beta_{6,i} MOM + \varepsilon_i \quad (23)$$

The results of the regression are displayed in Table I.

4.3 Technical Factors

In addition to estimating fundamental factor exposures, a set of technical indicators are calculated for model benchmarking. Technical indicators are factors and signals that are derived from the past values of the stock price itself. They are often used to analyze stock price patterns in order to make

Table I. ***Linear estimations of high-frequency fundamental factors exposures***

Regression β coefficients and the R^2 values for the rolling-window regressions over the entire data set for each stock and factor value. Each window consists of 100 consecutive 5-minute intervals, and values shown are averages per year ($*\ p \leq 0.1$ $**\ p \leq 0.05$ $***\ p \leq 0.01$).

PG	β Market	β SMB	β HML	β RMW	β CMA	β MOM	R^2
2013	0.846**	-0.307	-0.643	0.301	1.041	-0.003	39.6%
2014	0.731**	-0.178	-0.309	0.414	0.644	-0.353	35.3%
2015	0.840**	-0.178	-0.013	0.310	0.487	0.166	45.8%
2016	0.792**	-0.135	-0.184	0.207	0.714	0.485	38.8%
2017	0.525	-0.201	-0.189	0.203	0.344	-0.270	24.1%
Total	**0.747****	**-0.200**	**-0.268**	**0.287**	**0.646**	**0.005**	**36.7%**

XOM	β Market	β SMB	β HML	β RMW	β CMA	β MOM	R^2
2013	1.006***	-0.232	0.259	0.248	0.265	-0.435	48.5%
2014	1.114**	-0.335	0.459	-0.084	0.139	-0.321	47.9%
2015	0.916**	-0.333	0.174	-0.051	-0.075	-0.852**	57.5%
2016	0.918**	-0.344	0.359	-0.693	0.176	0.022	49.4%
2017	0.717**	-0.261	0.312	-0.836**	0.076	-0.491	47.4%
Total	**0.934****	**-0.301**	**0.312**	**-0.283**	**0.116**	**-0.416**	**50.1%**

NFLX	β Market	β SMB	β HML	β RMW	β CMA	β MOM	R^2
2013	1.372	-0.230	-0.974	-0.775	-0.916	0.483	28.9%
2014	1.150*	-0.433	-0.688	-0.785	-1.186	0.265	40.6%
2015	1.158*	-0.284	-0.330	-0.350	-1.287	-0.237	34.5%
2016	1.219*	-0.318	-0.408	-0.222	-1.047	-0.097	35.1%
2017	1.349*	-0.140	-0.773	-0.236	-0.336	0.475	35.3%
Total	**1.250***	**-0.281**	**-0.634**	**-0.474**	**-0.954**	**0.178**	**34.9%**

T	β Market	β SMB	β HML	β RMW	β CMA	β MOM	R^2
2013	0.866**	-0.303	-0.285	0.142	1.210	-0.390	35.9%
2014	0.805**	-0.174	0.033	0.292	0.314	-0.392	32.3%
2015	0.788**	-0.140	0.239	0.227	0.108	0.065	37.1%
2016	0.850**	-0.200	0.353	0.061	-0.028	0.537	35.5%
2017	0.820*	-0.248	0.434	0.200	-0.448	-0.678	26.3%
Total	**0.826**	**-0.213**	**0.155**	**0.184**	**0.231**	**-0.171**	**33.4%**

JPM	β Market	β SMB	β HML	β RMW	β CMA	β MOM	R^2
2013	1.243**	-0.176	1.695**	0.259	-0.307	0.435	52.7%
2014	1.294***	-0.079	0.803	-0.035	-0.006	-0.003	52.3%
2015	1.385***	-0.010	0.973**	-0.123	-0.055	0.382	63.4%
2016	1.240***	-0.107	0.969**	0.065	-0.380	-0.402	64.7%
2017	1.301***	-0.127	1.151**	0.017	-0.336	0.531	57.6%
Total	**1.292*****	**-0.100**	**1.118***	**0.036**	**-0.217**	**0.188**	**58.2%**

stock price predictions based on e.g., momentum or moving averages. For this paper, three technical indicators were selected:

- EMA(12) – exponential moving average with a window length of 12
- RSI(14) for a period 14 intervals
- %K – for a 3-period average of a %D estimated for 14 intervals

The technical indicators are similarly calculated on a rolling basis through the entire dataset for each stock.

4.4 Gramian Angular Field generation

The pictures of time series of 5-minute close prices, technical indicators and beta values for fundamental factors are converted into Gramian Angular Difference Fields. See Fig. 1 for illustration. As the input for the CNN on 5-minute intervals, we use a window size of 26 such intervals (and window size 20 for the daily intervals), graph these values for the close price and three factors, and convert these into four Gram matrices with the dimension of 26×26. The images are then labeled "LONG" or "SHORT" based on the subsequent rise or drop in stock price in the period following the last observation of the window. The model in this project will learn of the movements in stock price and the corresponding factors during the initial 1^{st} until t^{th} time step and make a prediction for the $t + 1^{st}$ time step. The target variable will be an indicator function based on the following criteria:

$$\text{target} = \begin{cases} 1, & \text{price}_{t+1} > \text{price}_t \\ 0, & \text{price}_{t+1} < \text{price}_t \end{cases} \qquad (24)$$

Given a target variable of 1, a "LONG" price recommendation is given, implying a long position in the asset is favorable during the interval between t and $t + 1$. If the target variable has a predicted value of 0, a "SHORT" price recommendation is given, implying a short position in the asset, alternatively not engaging in any position at all.

The input to the CNN is constructed by concatenating GADF images of certain factors in a 2×2 matrix of images, which results in a 52×52 matrix of pixels. For 20-day period, daily values of factors were aggregated using the mean and a rolling window of 20 days was used to create images that are 20×20 pixels. Each image that has been concatenated into a 2×2 matrix of images is labeled "LONG" or "SHORT" based on the subsequent close price. The exercise is to see if the pictures carry predictive information on the stock price when predicting its subsequent will rise or drop in price.

The generated images are stored in directories categorized as "LONG" with a target variable of 1 or "SHORT" with a target variable of 0, based on the condition explained in the previous paragraph. The full image dataset is then divided into a training set containing 70% of all images and a test set containing 15% of all images. A subsample of 17.65% of the train set is set aside for validation during training (15% of the whole dataset).

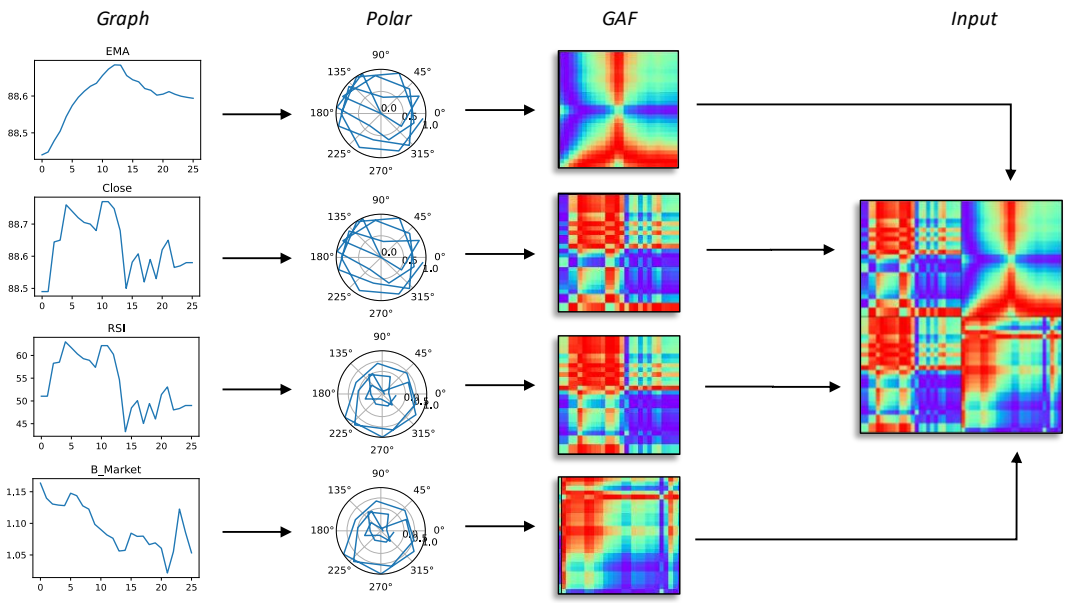


Figure 1. ***Converting close price and factor beta time series into GAF.*** Time series are first converted into polar coordinates and then transformed into a GAF, with the corresponding values rescaled to a RGB representation and each pixel corresponding to a value of the gram matrix. The input fed to the CNN network consists of four concatenated GAF images where each pixel represents the value of a factor or the close price at each unit of time. In the illustration, which is 26 consecutive 5-minute intervals, four images of pixel size 26×26 are concatenated into a single image with pixel dimensions of 52×52. In the case of 20-day intervals, the dimensions of a single image are 20 x 20.

4.5 CNN Configuration & Parameters

We create two different CNN for comparison across our trials. The chosen architectures are, at the core, inspired by conventional models provided by literature. Firstly, convolutional layers with square kernels are deployed with ReLU activation, followed by max pooling, in turn followed by batch normalization layers for better generalization and prevention of overfitting. The second model is similar to the first, but has additional features inspired by the CNNPred models of Hoseinzade & Haratizadeh (2019), who use kernel sizes that reflect the dimensions of the input images. The configurations and rationales behind the models are detailed below.

Single-Channel GAF 2D-CNN

As our baseline model, the Single-Channel GAF 2D-CNN is based on the classic LeNet-5 structure. It initially deploys three groups of layers, each containing a convolutional layer with ReLU activation, a max pooling layer and a batch normalization layer. The convolutional layers use 32 filters with kernel dimensions 3 x 3. This layer is followed by a max pooling operation of dimensions 2 x 2, thus reducing the feature maps to one fourth of the size of the input image. Finally comes a batch normalization layer. After this structure is repeated three times, a flattening layer follows to standardize output into a vector for the fully connected layers, in turn followed by a dropout layer to reduce model complexity with a hyperparameter of 0.5 for

the dropout rate. Following the dropout layer, there is a fully connected layer with 125 nodes, after which another dropout layer with a 0.3 dropout rate is used. Finally, an output layer is activated with two nodes and a softmax activation function. The output results in mutually exclusive probabilities for the likelihood of an image belonging to the LONG or SHORT class. See Fig. 2 for an illustration of this architecture.

Triple-Channel GAF 2D-CNN

An alternate, novel configuration that will be used is what we call the Triple-Channel GAF 2D-CNN. Unlike our single-channel baseline model, which extracts only one feature map from input GAF images through a square 3 x 3 kernel, this architecture extracts three feature maps from the images; once with a 3 x 3 square kernel, once with a horizontal vector kernel that slides downwards, and once with a vertical vector kernel that slides rightwards. The horizontal vector is of dimensions 26 x 1 or 20 x 1, depending on if we are using 5-minute or 20-day images. Similarly, the vertical vector is of dimensions 1 x 26 or 1 x 20. As such, they are of the same height or width as half the input images themselves. Note that vectors use strides of the same dimensions as themselves, meaning they only cover each pixel *once* during their convolutional operation. The 3 x 3 kernel on the other hand, has strides 1 x 1 and therefore slides over each pixel nine times, and generates output feature maps of the same dimensions as its input feature map (or input image in the case of the first layer). The middle channel of the architecture, as shown in Fig. 3, which uses the 3 x 3 kernels, is identical in design to the Single-Channel GAF 2D-CNN architecture except for the removal of batch normalization and dropout layers. The similar design to the single-channel CNN is intentional as we can then measure the marginal contribution of the additional vertical and horizontal channels. Finally, the output of all three channels is flattened and concatenated into a single vector that is used as input to a fully connected layer followed by a softmax-activated output layer which predicts whether the stock will appreciate or depreciate at $t + 1$.

We hope the Triple-Channel GAF 2D-CNN can capture additional spatial information that is unique to each of the four factors in the GAF input image. This is done in two ways: (1) the height of the vertical filter and the width of the horizontal filter are exactly the respective length and width of each pane in the GAF image, where each pane represents the time series of the factor betas or the closing price (see Fig. 1), which together with (2) having kernel strides set to their own dimensions ensures each filter never touches two panes at any given time. As a result, each pixel that these convolutional operations generate to form feature maps 3.1 and 2.1 in Fig. 3 can only come from a single pane, and exactly one fourth of the feature maps' pixels come from each pane.

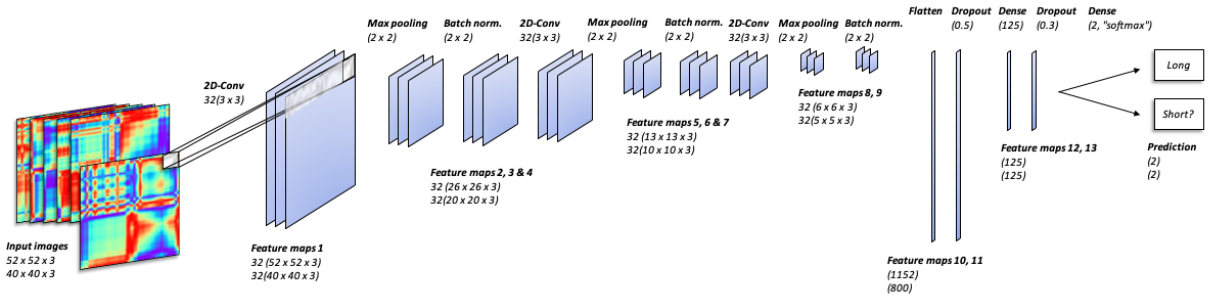


Figure 2. ***Single-Channel GAF 2D-CNN.*** The architecture is a variation of the classical LeNet-5 architecture, and consists of fourteen layers. First, three repetitions of the following layers are employed: a 2D convolutional layer with 32 filters of dimensions 3 x 3 is followed by a max pooling layer of dimension 2 x 2, which reduces the size of the convoluted images to a fourth, and lastly a batch normalization layer for regularization purposes. Second comes a flattening layer, followed by dropout layer, a fully connected layer with 125 nodes, another dropout layer, and finally an output layer with two nodes which is activated by a softmax function.

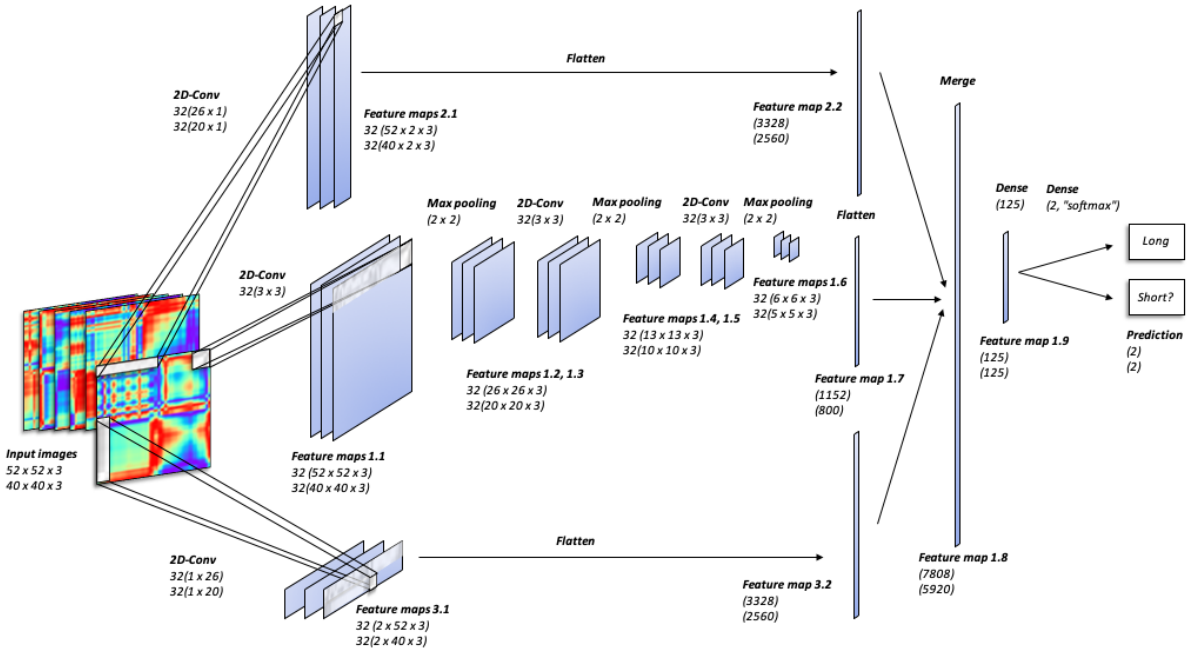


Figure 3. ***Triple-Channel GAF 2D-CNN.*** This architecture is constructed in a way similar to the Single-Channel GAF 2D-CNN, but has two additional channels that extract features from the input, no batch normalization layers and no dropout layers towards the end. The top channel is composed of a convolutional layer with 32 filters of dimensions 26 x 1 or 20 x 1, depending on the data set being used, followed by a flattening layer. The bottom channel is identical but horizontally oriented, consisting of a convolutional layer with 32 filters of dimensions 1 x 26 and 1 x 20, followed by a flattening layer. The intermediary output of all three channels is then concatenated and followed by two fully connected layers.

Loss Function

The loss function is an integral part of any machine learning algorithm. Specifying a correct loss function is necessary for efficient optimization of the kernel weights and yielding accurate results. The loss function is used to calculate the gradient and subsequently update the weights of the model in the direction opposite of the gradient. This process is repeated until the algorithm reaches the minimum of the loss function, or alternatively when the incremental improvement in loss is below a certain threshold. Given the nature of the classification task, the selected loss function is categorical cross entropy loss with two classes.

$$CE = -\sum_{i=1}^{C=2} t_i \, log\big(f(s_i)\big)$$

$$= -t_1 log(f(s_1)) - (1 - t_1) \, log\big(1 - f(s_1)\big) \quad (25)$$

$f(s_1)$ and $1 - f(s_1)$ are the scores from the output of the CNN passed through an activation function and t_1 , $1 - t_1$ are the true binary value of the target variable.

Optimizer

One of the most common optimizers of neural networks is the Stochastic Gradient Descent (SGD) method. A gradient descent optimizer for a given neural network architecture is initiated on a given loss function that is dependent on the weights of kernels and operators in a neural network. The partial derivative of the gradient with respect to the weight is calculated, and the weights are updated in the opposite direction of the gradient in a higher dimensional space, consequently reducing the error of the model and improving its performance. Vanilla SGDs may encounter convergence issues near local minima, where the gradient is almost perpendicular to the minima and oscillates closely, thus taking a long time to reach the extreme point. Momentum introduces an inertia parameter that helps smooth out oscillations, thus drastically improving the speed of convergence and thus reducing the time needed to find the extreme point. An SGD optimizer with a momentum contribution has been employed in this paper. Derivations of the function can be found in the works of Ning Qian (Qian, 1999) and Sebastian Ruder (Ruder, 2016).

4.6 Model Evaluation Metrics

The primary metrics for evaluations of our classification task are based on confusion matrices and their corresponding metrics that are associated with it. The confusion matrix is an array of true classes and predicted classes, which shows how the distributed predictions perform against the true values. An illustration of a confusion matrix is provided in Fig. 4, with its associated terms.

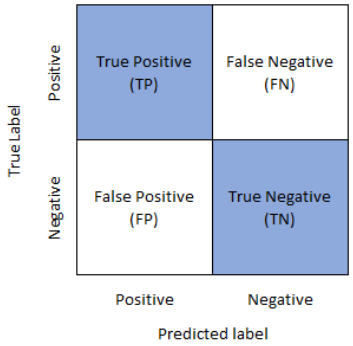


Figure 4. ***Confusion matrix illustration.*** Correctly predicted classes are along the main diagonal.

Accuracy is defined as the number of correct predictions divided by the total number of

predictions. In our case, it gives an indication of how accurate the algorithm is at correctly predicting the movements of a stock if all classes are equally important.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (26)$$

Precision and recall are two other metrics that can be used in conjunction with accuracy. Precision gives a measure of how often something that is positive is predicted as positive. In this case, it tells us how many times we are correct in our prediction of a long position in an asset. The recall measure tells us the percentage of positives that were predicted as the positive class.

$$Precision = \frac{TP}{TP + FP} \quad (27)$$

$$Recall = \frac{TP}{TP + FN} \quad (28)$$

F_1 Score is a harmonic mean of precision and recall and as such accounts for both false positives and false negatives. F_1 Score is used together with the accuracy to gauge performance of the model, and is especially useful if there is a sample imbalance, e.g. many more positive class occurrences than negative.

$$F_1 = \frac{2TP}{2TP + FP + FN} \quad (29)$$

5. Results

The two models, Single-Channel GAF 2D-CNN and Triple-Channel GAF 2D-CNN were trained and evaluated on six data sets of images. The first three data sets consist of GAF images of 5-minute interval data that represent four factors of different characteristics: technical indicators, fundamental factors, and a combination of the two:

1. Technical: Close, EMA, %K, RSI
2. Fundamental: Close, Market, SMB, HML
3. Combination: Close, Market, HML, RSI

The latter three datasets consist of images composed of mean-aggregated 20-day interval data, again with different factor betas depending on their characteristics:

1. Technical: Close, EMA, %K, RSI
2. Fundamental: Close, Market, SMB, HML.
3. Combination: Close, Market, HML, RSI

The 5-minute data sets consist of a total of 18,795 GAF images, of which 9,049 are labeled “LONG” and 9,746 “SHORT” depending on the subsequent stock price movement. The data can thus be considered balanced across the two classes. The pictures are thereafter divided into a train, validation and test set, corresponding to 70%, 15% and 15% of the original dataset respectively. The 20-day data sets consist of a total of 6,180 images with the same train, validation test split. During training, the data set is split into randomized batches of size 32 during each trial. Running the two models on the six datasets yields 12 different specifications.

The number of epochs specified is the number of times the model trains on the entire data set. Between each epoch, an evaluation is done on the validation data set enabling monitoring of the training process and generalization of the model. Each model is trained on each dataset for five trials, after

Table II. ***Average performance metrics of the trained CNN architectures***

Single-Channel and Triple-Channel GAF 2D-CNN were trained, validated and tested on each of the six data sets: per 5-minute and 20-day frequency and per factor composition on each image (fundamental, technical and a combination of the two). In total, 60 trials were completed with 20 epochs for training and batch sizes of 32. Performance metrics such as Precision, Recall, F1-score, Accuracy and the test-set loss are reported. Detailed metrics for each trial are presented in Appendix B.

Trial specifications			**Test performance**					**Train Performance**	
Time frame	Factors	CNN model	Avg. Precision	Avg. Recall	Avg. F1	Avg. Accuracy	Avg. Loss	Avg. Accuracy	Avg. Loss
5 min	Fundamental	Single	0.498	0.501	0.573	0.501	0.717	0.592	0.691
		Triple	0.497	0.499	0.540	0.499	0.658	0.586	0.677
	Technical	Single	0.504	0.515	0.641	0.515	0.714	0.546	0.706
		Triple	0.509	0.512	0.594	0.512	0.703	0.549	0.688
	Combination	Single	0.501	0.507	0.581	0.507	0.711	0.589	0.714
		Triple	0.509	0.511	0.561	0.511	0.701	0.604	0.668
20 day	Fundamental	Single	0.508	0.510	0.473	0.510	0.922	0.681	0.773
		Triple	0.508	0.507	0.500	0.507	0.690	0.701	0.617
	Technical	Single	0.499	0.499	0.450	0.499	0.846	0.629	0.771
		Triple	0.506	0.503	0.513	0.503	0.776	0.626	0.671
	Combination	Single	0.484	0.483	0.434	0.483	0.906	0.699	0.773
		Triple	0.539	0.536	0.524	**0.536**	0.691	0.686	0.655

which the results are aggregated and averaged. Results are displayed in Table II (see Appendix B for individual results in each trial). Among the trained models, the best predictive performance was achieved with the Triple-Channel GAF 2D-CNN on the 20-day dataset with a combination of fundamental and technical factors (interestingly, this is also where the Single-Channel GAF 2D-CNN performed the worst). Average test accuracy was 53.6%, with a training accuracy of 68.6%. Over the 20 epochs during training and across the five trials, we saw continuous improvement in training accuracy (see Appendix A), gradual decrease of training and validation loss, and above 50% test accuracy for each of the five trials. The Single-Channel GAF 2D-CNN

model performed best on the 5-minute data set with technical factors, with an average accuracy of 51.5% and training accuracy of 54.6% during the five trials. The results of this model are skewed towards "SHORT" predictions, as can be seen in the confusion matrix of this trial in Appendix A. This is a symptom of the model not being able to successfully extract relevant features and generalize to the test set. The model however had consistently high F-1 scores, as well as precision and recall values. The Triple-Channel GAF 2D-CNN achieved similar results on this dataset with less skewed results.

Both models performed poorly on the 5-minute fundamental factors dataset, where Single-Channel GAF 2D-CNN and Triple-Channel GAF 2D-CNN achieved 50.1% and 49.9% accuracy respectively. Despite the training accuracy of approximately 60% and low loss scores on both the validation and train set, the models performed poorly on test sets. Both models achieved above average results on the 20-day fundamental factor dataset, with training accuracies around 70% and test accuracy of 51%.

Conversely, we find that both architectures perform better with technical factors when predicting stock returns within minutes rather than after 20 days. For the 5-minute data set with technical factors, Single-Channel GAF 2D-CNN and Triple-Channel GAF 2D-CNN achieved test accuracies of 51.5% and 51.2% and train accuracies of 54.6% and 54.9% respectively.

For the data sets with a combination of technical and fundamental factors (Close, Market, HML, RSI), the results are rather inconclusive. On the 5-minute data sets, the two architectures performed better than with only fundamental factors, but worse than when using only technical factors. On the 20-day data sets, the Single-Channel CNN performed worse than on any other data set, while the Triple-Channel CNN performed remarkably better than on any other data set.

In general, the achieved training accuracies were much higher on the 20-day data sets (between 59.4% and 73.3% for all trials and both models) compared to 5-minute data sets (between 52.3% and 61.7%). At the same time, test accuracies did not improve but became more dispersed for 20-day data sets (average 50.6%, standard deviation 1.89%) compared to 5-minute data sets (average 50.7%, standard deviation 0.82%).

6. Discussion

With regards to the different factor betas, it still seems fundamental factors are better at predicting stock returns in the longer term, relative to technical factors. Similarly, technical factors seem fit for predicting returns in the short term. This is in line with classical finance theory and the way the community model returns in each situation today. With regards to the two models, we found them to perform similarly across the different data sets except for the 20-day data set with a combination of fundamental and technical factors, where the Triple-Channel GAF 2D-CNN achieved tests accuracies consistently above 53%. Speculatively, the superiority of the triple-channel model could stem from the additional spatial information that the three, differently sized input kernels can extract. The combination of technical and

fundamental factors could arguably provide further spatiality, thus yielding better results. This shows that it is worth exploring CNN architectures and inputs outside of the conventional formats provided in literature.

In general, the results did not show distinctive predictive performance among any data sets, but some trends emerged during different trials. One is that fundamental factors showed very little predictive power in the classification task, achieving average to slightly above average performance from the two models compared to simply guessing (would correspond to 50% accuracy). There are some possible explanations to why the models were not able to extract useful features from images or generalize these well to the validation and test sets.

A methodological procedure that may have interfered with the exercise is the initial factor construction. We employed rolling linear regressions on five-minute intervals with a sample size of 100 for each factor, for each regression. We obtained widely varying R^2 estimates (from 24% to 65%) and low significance levels for our beta estimates, besides for the Market factor. The scope was not like that of classical finance, i.e. to see if the factor returns provided some form of systemic, linear risk exposure to a given asset, but rather to see if there are non-linear relations between factor movements and the given stocks' returns in the short term via deep learning. In the classical setting, we could conclude that we cannot be certain that the factors provide any systemic risk exposures, as they have shown low significance. However, in our setting, we obtain noisy beta estimates that we use for our exercise, and thus may use only noise as our "factor betas". Some of this statistical noise is then learnt by the convolutional neural networks during training, resulting in worse generalization over the validation and test set.

In this paper, we have utilized a classification criterion based on the directly subsequent period for classifying a picturized time series as "LONG" or "SHORT". Although this is the most intuitive approach, it is worth considering longer forecasting horizons. Shynkevich et. al. (2017) suggests using the same forecast period as the window length for optimal results. This could mitigate the issue that arises from classifying images on the usually very minor price fluctuations observed on 5-minute stock data. For example, if a stock is trading at USD 74.14 at 10:15 and then is trading at USD 74.15 at 10:20, the original classification criteria would label the data as "LONG", although the price increase is insignificant at ~0.01% and is likely just noise. This issue could also be resolved by using a multi-class classification task as opposed to the binary classification proposed in this paper, namely a "LONG", "HOLD", "SHORT" split. This could potentially improve predictive performance by setting higher absolute thresholds for stock returns to be labelled "LONG" or "SHORT", and thus filing uninformative images into "HOLD".

Two issues encountered during trials include (1) the algorithm labeling all data points as belonging to a single class, either all "SHORT" or all "LONG", which can be seen in confusion matrices in Appendix B, and (2)

the inability of CNN to generalize training accuracy to validation and testing. Although training accuracy consistently grew over epochs (Appendix A) to accuracies of 60 to 70%, and both training and validation loss decreased, any additional epochs that allowed the model to train to above 70% accuracy resulted in drastic increases in validation loss, implying severe overfitting. We saw only marginal improvement when adjusting for overfitting by employing pooling, batch normalization and dropout layers. The aforementioned implies that the model either learns well on the training set but is not able to generalize well to the test set (overfitting), or that the statistical noise in the data sets inhibit the models from learning informative features to make accurate classifications on the test set.

7. Conclusion

In this paper, two convolutional neural networks have been developed and trained on three different datasets of input features, at two data frequencies, in an attempt to classify equity time series into "LONG" and "SHORT" categories based on price appreciation or depreciation on a high-frequency interval. The data sets were constructed by estimating high-frequency fundamental factor exposures of selected equities over a 5-year period and calculating technical indicators on a high-frequency level for each stock. Three datasets of GAF images were constructed for each type of factors – one for fundamental factors, one for technical factors, and one as a combination of the two, where four GAF images were concatenated into a 2 x 2 matrix of images that were used as input for the convolutional neural networks.

The proposed CNN models improved relatively well on training data, reaching accuracies of 60 to 70%, and significantly decreasing the loss on both the training and validation set, however, the models were not able to generalize well to the test set, resulting in accuracies some percentage points above 50%, implying marginally better results than random guessing.

The models trained on the 20-day fundamental factors performed better than the models trained on 5-minute fundamental factors, supporting the consensus that fundamental factors are primarily useful for long term return forecasting. Conversely, technical factors obtained better performance on the 5-minute frequency than on the 20-day frequency, as measured by the test set accuracy. The results of the combination data set of technical and fundamental factors remain inconclusive as the two models achieved contradicting results.

For further research we would suggest using alternative methods for estimating the fundamental factor exposures at high frequencies to ensure correct beta estimates are used. As it currently is, noise in the estimates is internalized by the convolutional network, thus distorting our results. Furthermore, we would suggest using a higher percentage threshold for classifying time series, as some of the images classified as "LONG" or "SHORT" only saw movements in the closing price of approimately 0.01%.

References

Abe, M., & Nakamaya, H. (2018). Deep Learning for Forecasting Stock Returns in the Cross-Section . *Advances in Knowledge Discovery and Data Mining*, 273-284.

Aït-Sahalia, Y., Kalnina, I., & Xiu, D. (2020). High-frequency factor models and regressions. *Journal of Econometrics*, 86–105.

Brock, W., Lakonishok, J., & LeBaron, B. (1992). *Simple Technical Trading Rules and the Stochastic Properties of Stock Returns* (Vol. 45(5)).

Cai, S., Feng, X., Deng, Z., Ming, Z., & Shan, Z. (2018). Financial news quantization and stock market forecast research based on CNN and LSTM. *International Conference on Smart Computing and Communication,.* Tokyo: Springer: Berlin/Heidelberg.

Carhart, M. M. (1997). On persistance in mutual fund performance. *The Journal of Finance, 52*(1), 57-82.

Chen, J.-H., & Tsai, Y.-C. (2020). Encoding candlesticks as images for pattern classification using convolutional neural networks. Taipei: arXiv:1901.05237v2 [cs.CE].

Chen, S., & Ge, L. (2019). Exploring the attention mechanism in LSTM-based Hong Kong stock price movement prediction . *Quantitative Finance*, 1507–1515.

Di Persio, L., & Honchar, O. (2016). Artificial Neural Networks architectures for stock price prediction: comparisons and applications. *International Journal of Circuits, Systems and Signal Processing*, 403-413.

Ding, X., Zhang, Y., Liu, T., & Duan, J. (2015). Deep Learning for Event-Driven Stock Prediction. *Twenty-Fourth International Joint Conference on Artificial Intelligence.* Buenos Aires.

Enke, D., & Zhong, X. (2016). Forecasting daily stock market return using dimensionality reduction. *Expert Systems With Applications*, 126-139.

Fama, E. (1970). Efficient Capital Markets: A Review of Theory and Empirical Work. *Journal of Finance, 25*(2), 383-417.

Fama, E. F., & French, K. R. (1993). Common risk factors in the returns on stocks and bonds. *Journal of Financial Economics, 33*, 3-56.

Fama, E. F., & French, K. R. (2015). A five-factor asset pricing model. *Journal of Financial Economics, 116*, 1-22.

Fan, J., Xue, L., & Yao, J. (2017). Sufficient forecasting using factor models. *Journal of Econometrics, 201*, 292-306.

Feng, G., Giglio, S., & Xiu, D. (2020). Taming the Factor Zoo: A Test of New Factors. *Journal of Finance, 75*(3), 1327-1370.

Feng, G., Polson, N. G., & Xu, J. (2019). Deep Learning in Characteristics-Sorted Factor Models. arXiv:1805.01104 Help | Advanced Search .

Fukushima, K. (1980). *Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position.* Tokyo: NHK Broadcasting Science Research Laboratories.

Gu, J., Kuen, J., Ma, L., Shahroudy, A., Shuai, B., & Chen, T. (2018). Recent advances in convolutional neural networks. *Pattern Recognition, 77*, 354-377.

Gudelek, U., Ozbayoglu, M., & Boluk, A. (2017). A deep learning based stock trading model with 2-D CNN trend detection. *IEEE Symposium Series on Computational Intelligence (SSCI).* ResearchGate.

Gunduz, H., Yaslan, Y., & Cataltepe, Z. (2017). Intraday prediction of Borsa Istanbul using convolutional neural networks and feature correlations. *Knowledge-Based Systems*, 138–148.

Hoseinzade, E., & Haratizadeh, S. (2019). CNNpred: CNN-based stock market prediction using a diverse set of variables. *Expert Systems With Applications*, 273–285.

Hu, Z., Zhao, Y., & Khushi, M. (2021). A Survey of Forex and Stock Price Prediction Using Deep Learning. *Applied System Innovation*, 1-30.

Kara, Y., Boyacioglu, M. A., & Baykan, Ö. K. (2011). Predicting direction of stock price index movement using artificial neural networks and support vector machines: The sample of the Istanbul Stock Exchange. *Expert Systems with Applications*, 5311–5319.

Kraus, M., & Feuerriegel, S. (2017). Decision support from financial disclosures with deep neural networks and transfer learning. *Decision Support Systems, 104*, 38-48.

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R., Hubbard, W., & Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. *Neural computation, 1*(44), 541-551.

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-Based Learning Applied to Document Recognition. *Institute of Electrical and Electronics Engineers.* New Jersey.

Lindsay, G. W. (2020). Convolutional Neural Networks as a Model of the Visual System: Past, Present, and Future. *Journal of Cognitive Neuroscience.*, 1-15.

Liu, S., Zhang, C., & Ma, J. (2017). CNN-LSTM Neural Network Model for Quantitative Strategy Analysis in Stock Markets. *International Conference on Neural Information Processing.* Guangzhou: Springer:Berlin/Heidelberg,.

Liu, Y., Zeng, Q., Yang, H., & Carrio, A. (27–28 August 2018). Stock price movement prediction from financial news with deep learning and knowledge graph embedding. *Pacific Rim Knowledge Acquisition Workshop.* Nanjing.

Long, W., Lu, Z., & Cui, L. (2018). Deep learning-based feature engineering for stock price movement prediction. *Knowledge-Based Systems*, 163-173.

Maqsood, H., Mehmood, I., Maqsood, M., Yasir, M., Afzal, S., Aadil, F., . . . Muhammad, K. (2020). A local and global event sentiment based efficient stock exchange forecasting using deep learning. *International Journal of Information Management*, 432-451.

Ozbayoglu, A. M., & Sezer, O. B. (2018). Algorithmic financial trading with deep convolutional neural networks: Time series to image conversion approach. *Applied Soft Computing*, 525–538.

Patel, J., Shah, S., Thakkar, P., & Kotecha, K. (2015). redicting stock and stock price index movement using Trend Deterministic Data Preparation and machine learning techniques. *Expert Systems with Applications*, 259–268.

Qian, N. (1999). On the Momentum Term in Gradient Descent Learning. *12*(1), 145-151.

Ruder, S. (2016). An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*.

Selvin, S., Ravi, V., Gopalakrishnan, E. A., & Menon, V. K. (2017). Stock price prediction using LSTM, RNN and CNN-sliding window model. *017 International Conference on Advances in Computing, Communicationsand Informatics (ICACCI)*, . Manipal.

Sezer, O. B., Gudelek, M. U., & Ozbayoglu, A. M. (2020). Financial time series forecasting with deep learning : A systematic literature review: 2005–2019. *Applied Soft Computing Journal, 90*, 106181.

Sharpe, W. F. (1964). Capital asset prices: a theory of market equilibrium under conditions of risk. *Journal of Finance, 19*(3), 425-442.

Shynkevich, Y., McGinnity, T. M., Coleman, S. A., Belatreche, A., & Li, Y. (2017). Forecasting price movements using technical indicators: Investigating the impact of varying input window length. *Neurocomputing, 264*, 71-88.

Sim, H. S., Kim, H. I., & Ahn, J. J. (2019). Is Deep Learning for Image Recognition Applicable to Stock Market Prediction? *Complexity*, 0-10.

Wang, Z., & Oates, T. (2015). Encoding Time Series as Images for Visual Inspection and Classification Using Tiled Convolutional Neural Networks. University of Maryland Baltimore County.

Yang, C., Zhai, J., & Tao, G. (2020). Deep Learning for Price Movement Prediction Using Convolutional Neural Network and Long Short-Term Memory. *Mathematical Problems in Engineering*, 0-13.

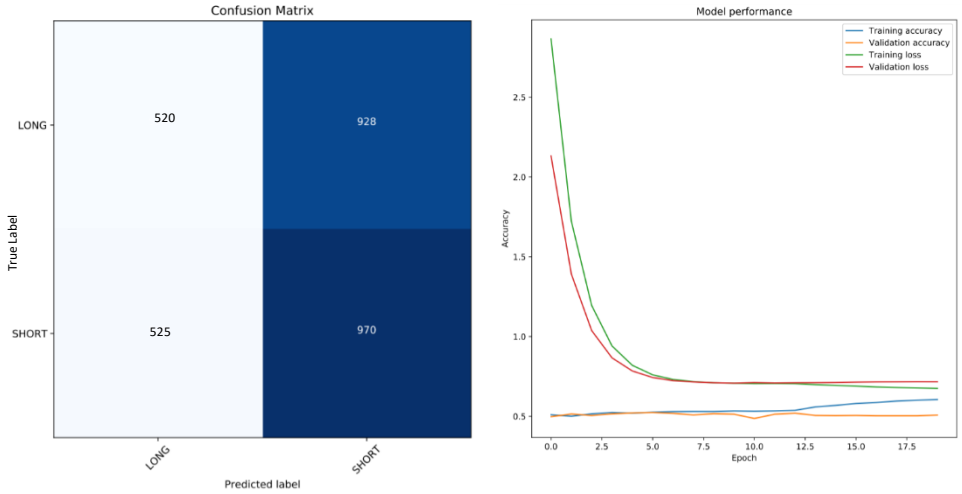
Yang, H., Zhu, Y., & Huang, Q. (2018). A multi-indicator feature selection for cnn-driven stock index prediction. *International Conference on Neural Informa- tion Processing* (pp. 35-46). Berlin: Springer.

Zhou, B. (2019). Deep learning and the cross-section of stock returns: Neural networks combining price and fundamental information. SSRN Electron.

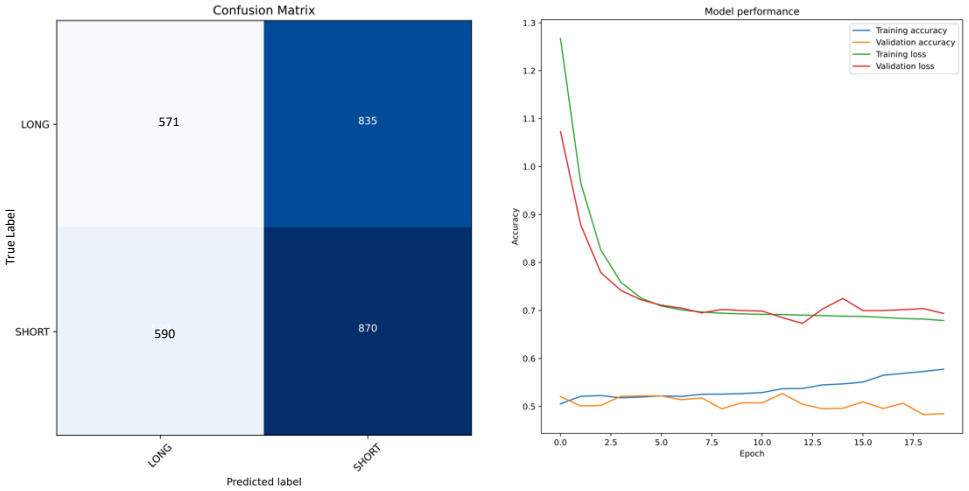
Appendix A – Best performing trial confusion matrix and plot of accuracy and loss over train and validation set

5-min fundamental factor dataset

Single-Channel GAF 2D-CNN

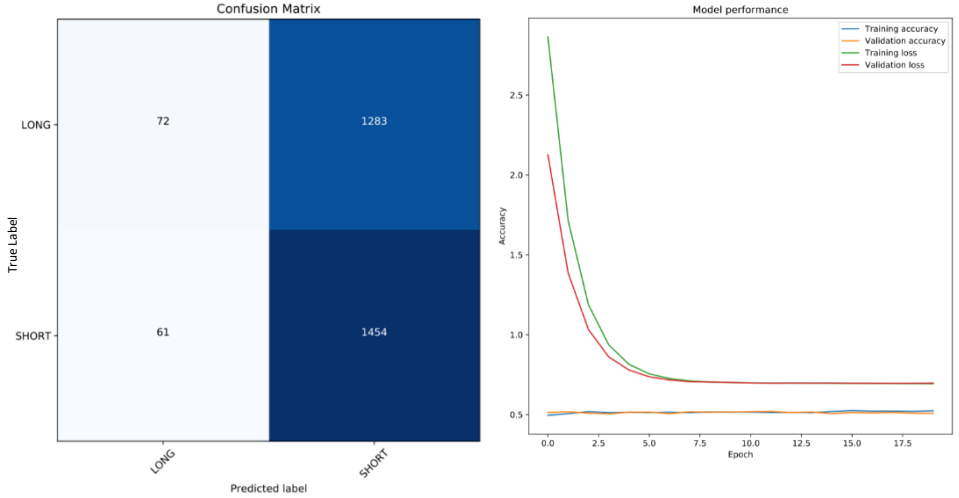


Triple-Channel GAF 2D-CNN

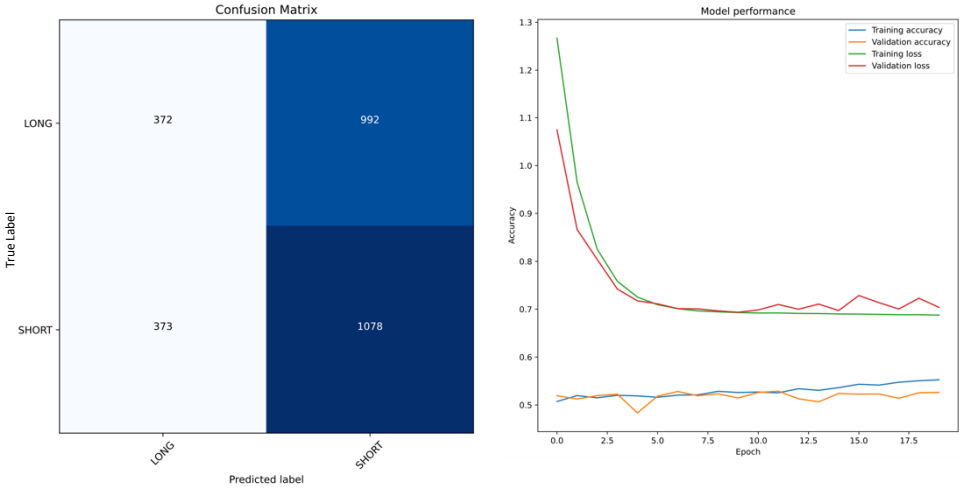


5-minute technical indicators dataset

Single-Channel GAF 2D-CNN

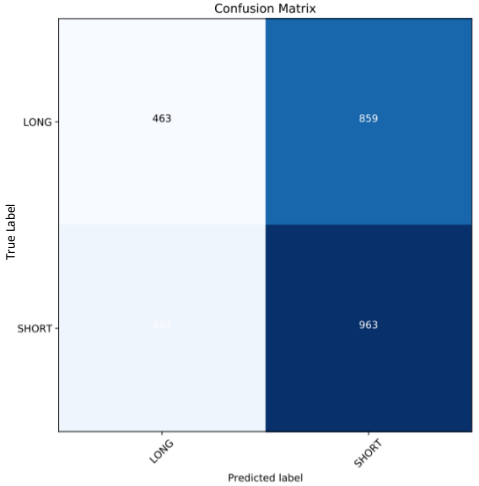


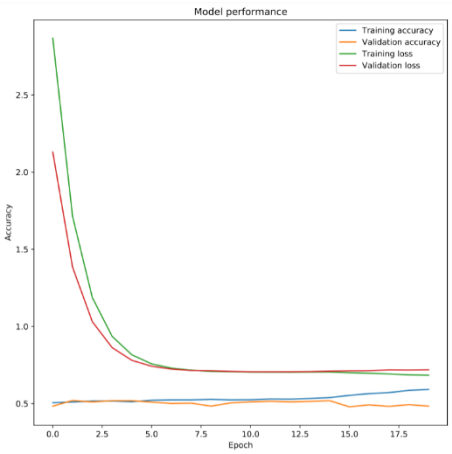
Triple-Channel GAF 2D-CNN



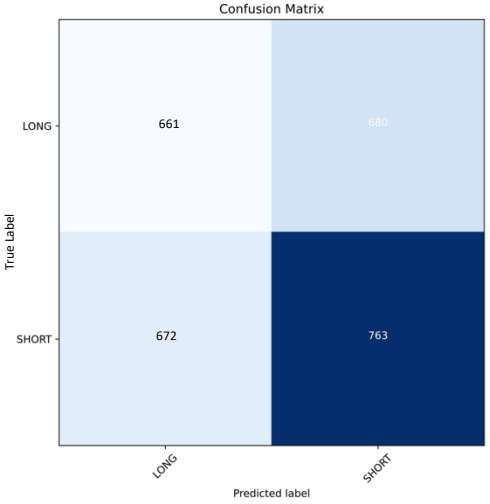
5-min combination factor dataset

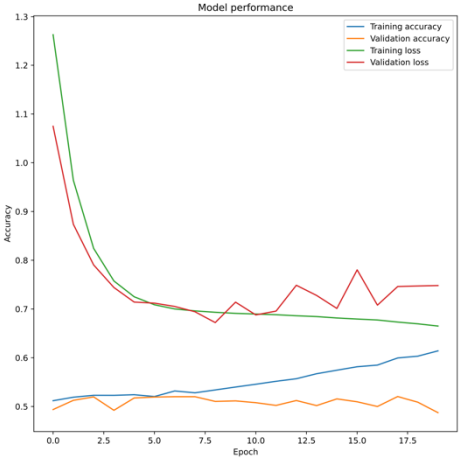
Single-Channel GAF 2D-CNN





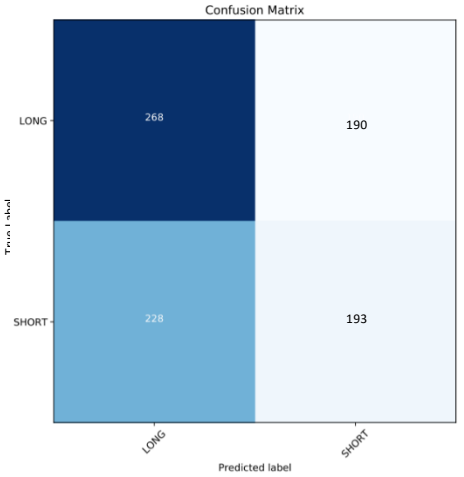
Triple-Channel GAF 2D-CNN

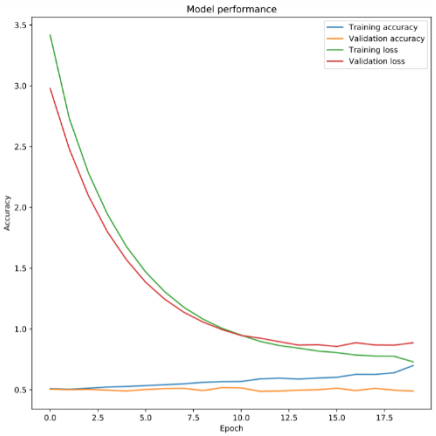




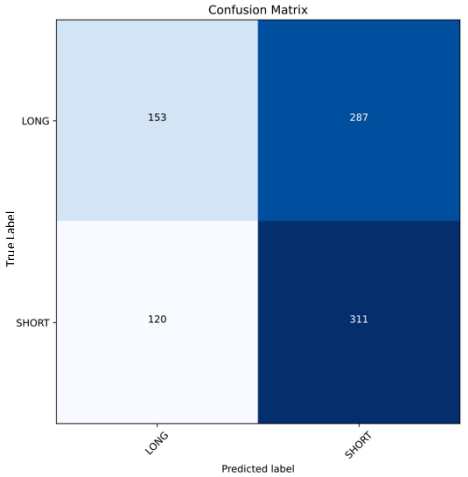
20-day fundamental factors dataset

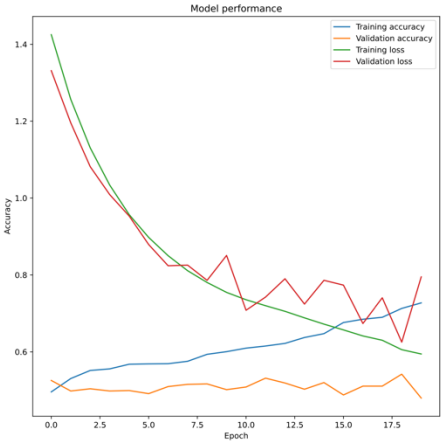
Single-Channel GAF 2D-CNN





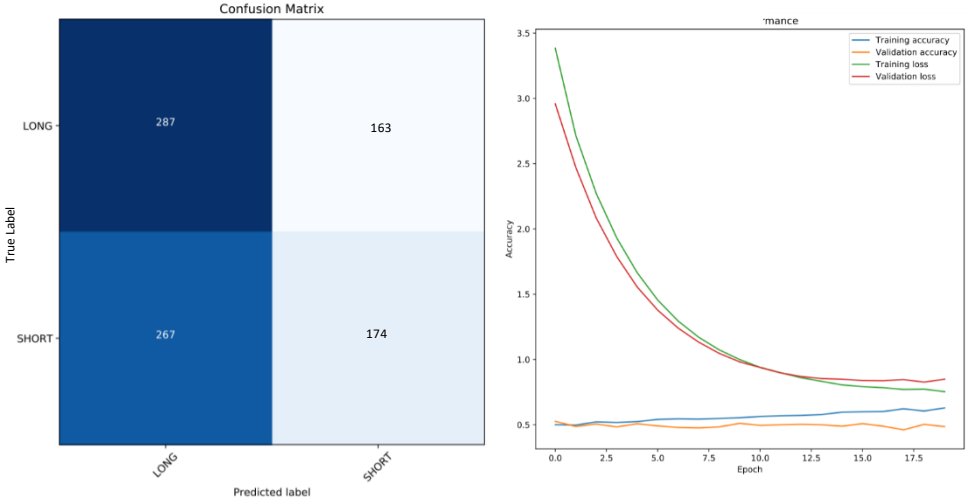
Triple-Channel GAF 2D-CNN





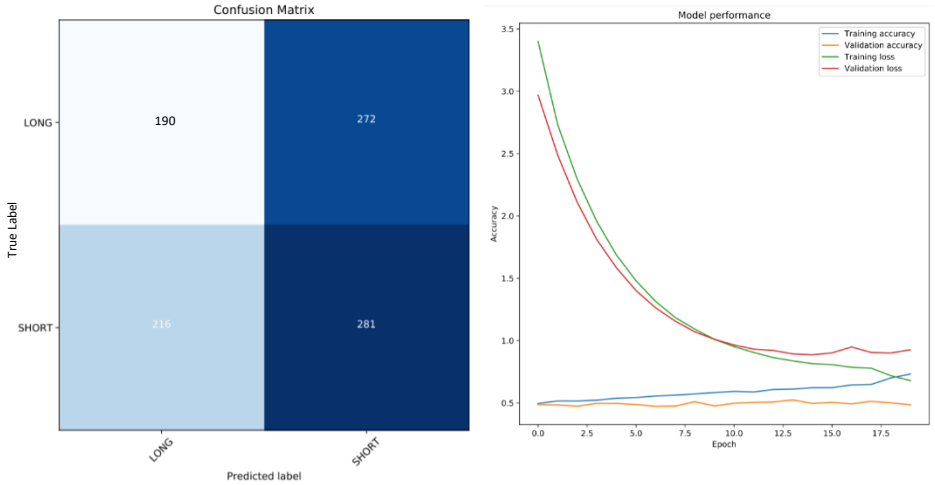
20-day technical indicator dataset

Single-Channel GAF 2D-CNN

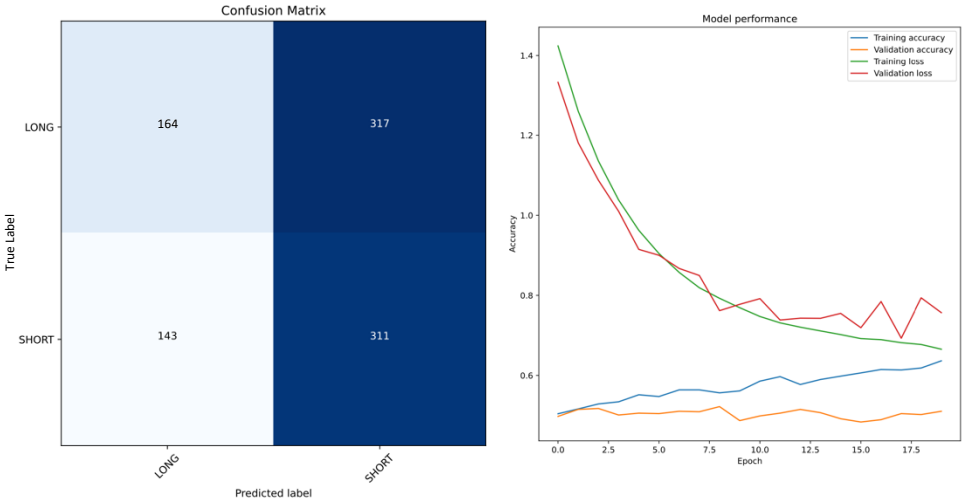


20-day combination factor dataset

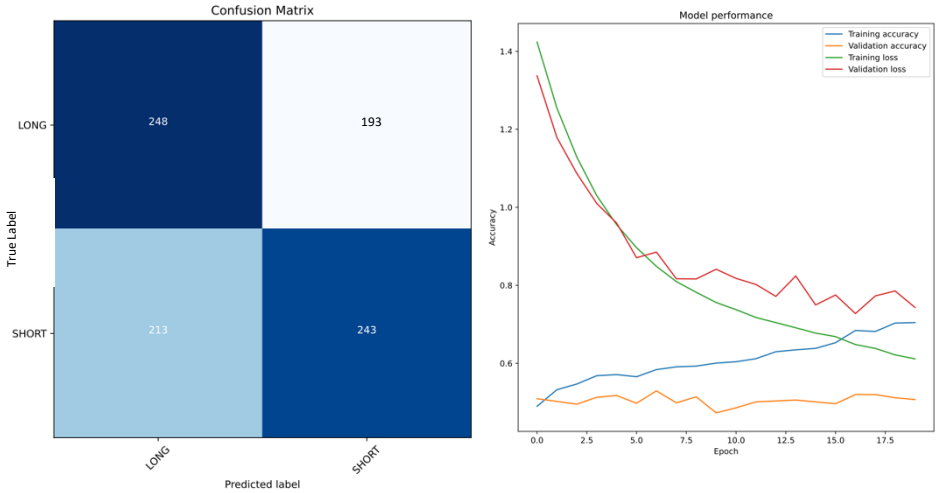
Single-Channel GAF 2D-CNN



Triple-Channel GAF 2D-CNN



Triple-Channel GAF 2D-CNN



Appendix B – Results from model training and evaluation

Trial specifications				Test performance					Train performance		Parameters				
Time frame	Factors	Architecture	Trial #	Precision	Recall	F1	Accuracy	Loss	Accuracy	Loss	Train #	Validation #	Test #	Epochs #	CNN # of parameters
5 min	Fundamental	Single-Channel	1	0.501	0.504	0.589	0.504	0.711	0.578	0.688	13 053	2 797	2 943	20	163 961
		GAF 2D-CNN	2	0.490	0.493	0.550	0.493	0.723	0.596	0.682					
			3	0.504	0.506	0.572	**0.506**	0.721	**0.605**	0.717					
			4	0.497	0.501	0.580	0.501	0.713	0.585	0.686					
			5	0.496	0.499	0.572	0.499	0.717	0.594	0.682					
5 min	Fundamental	Triple-Channel	1	0.499	0.499	0.514	0.499	0.616	**0.601**	0.673	13 220	2 707	2 866	20	1 000 825
		GAF 2D-CNN	2	0.500	0.500	0.511	0.500	0.687	0.581	0.678					
			3	0.501	0.503	0.550	**0.503**	0.671	0.578	0.679					
			4	0.488	0.492	0.573	0.492	0.666	0.572	0.682					
			5	0.498	0.500	0.554	0.500	0.650	0.598	0.673					
5 min	Technical	Single-Channel	1	0.536	0.532	0.684	**0.532**	0.695	0.523	0.693	13 114	2 809	2 870	20	163 961
		GAF 2D-CNN	2	0.496	0.511	0.622	0.511	0.732	0.558	0.715					
			3	0.495	0.505	0.590	0.505	0.727	**0.563**	0.714					
			4	0.515	0.518	0.661	0.518	0.694	0.534	0.694					
			5	0.480	0.509	0.646	0.509	0.723	0.554	0.713					
5 min	Technical	Triple-Channel	1	0.498	0.506	0.614	0.506	0.706	0.551	0.688	13 263	2 715	2 815	20	1 000 825
		GAF 2D-CNN	2	0.510	0.515	0.612	**0.515**	0.703	0.547	0.689					
			3	0.510	0.515	0.612	**0.515**	0.716	0.553	0.688					
			4	0.516	0.513	0.523	0.513	0.675	**0.553**	0.687					
			5	0.508	0.513	0.609	0.513	0.716	0.543	0.689					

5 min	Combination	Single-Channel GAF 2D-CNN	1	0.506	0.513	0.592	0.513	0.713	0.592	0.712	13 197	2 827	2 769	20	163 961
			2	0.488	0.494	0.559	0.494	0.710	**0.594**	0.715					
			3	0.498	0.504	0.578	0.504	0.710	0.589	0.713					
			4	0.502	0.508	0.586	0.508	0.707	0.576	0.712					
			5	0.510	0.515	0.589	**0.515**	0.715	0.592	0.719					
5 min	Combination	Triple-Channel GAF 2D-CNN	1	0.513	0.513	0.530	**0.513**	0.681	0.614	0.665	13 295	2 722	2 776	20	1 000 825
			2	0.511	0.512	0.536	0.512	0.698	0.612	0.660					
			3	0.503	0.509	0.596	0.509	0.715	0.567	0.683					
			4	0.507	0.510	0.571	0.510	0.680	0.611	0.669					
			5	0.509	0.513	0.575	**0.513**	0.731	**0.617**	0.663					
20 days	Fundamental	Single-Channel GAF 2D-CNN	1	0.502	0.501	0.478	0.501	0.925	0.679	0.777	4 367	934	879	20	119 961
			2	0.504	0.511	0.380	0.511	0.894	0.659	0.785					
			3	0.510	0.510	0.527	0.510	0.935	**0.701**	0.747					
			4	0.503	0.502	0.497	0.502	0.965	0.669	0.794					
			5	0.523	0.525	0.480	**0.525**	0.890	0.699	0.761					
20 days	Fundamental	Triple-Channel GAF 2D-CNN	1	0.507	0.506	0.542	0.506	0.787	**0.733**	0.592	4 386	897	897	20	763 673
			2	0.540	0.533	0.605	**0.533**	0.760	0.727	0.594					
			3	0.500	0.506	0.355	0.506	0.356	0.688	0.629					
			4	0.496	0.494	0.512	0.494	0.353	0.674	0.641					
			5	0.498	0.497	0.489	0.497	1.195	0.684	0.631					

20 days	Technical	Single-Channel GAF 2D-CNN	1	0.486	0.493	0.376	0.493	0.868	0.619	0.764	4 356	933	891	20	119 961
			2	0.512	0.513	0.445	0.513	0.825	**0.662**	0.771					
			3	0.493	0.495	0.411	0.495	0.827	0.602	0.798					
			4	0.476	0.478	0.569	0.478	0.851	0.635	0.745					
			5	0.527	0.517	0.447	**0.517**	0.858	0.629	0.778					
20 days	Technical	Triple-Channel GAF 2D-CNN	1	0.494	0.487	0.617	0.487	0.782	0.594	0.684	4 355	890	935	20	763 673
			2	0.500	0.508	0.286	**0.508**	0.844	**0.644**	0.664					
			3	0.515	0.508	0.575	**0.508**	0.721	0.637	0.666					
			4	0.513	0.507	0.562	0.507	0.817	0.631	0.666					
			5	0.507	0.505	0.524	0.505	0.715	0.623	0.676					
20 days	Combination	Single-Channel GAF 2D-CNN	1	0.482	0.484	0.415	0.484	0.882	0.673	0.765	4 300	921	959	20	119 961
			2	0.474	0.472	0.461	0.472	0.921	0.710	0.789					
			3	0.489	0.491	0.535	**0.491**	0.913	**0.732**	0.756					
			4	0.488	0.484	0.422	0.484	0.904	0.725	0.774					
			5	0.488	0.482	0.338	0.482	0.910	0.657	0.781					
20 days	Combination	Triple-Channel GAF 2D-CNN	1	0.548	0.547	0.545	**0.547**	0.784	**0.704**	0.743	4 339	884	897	20	763 673
			2	0.541	0.536	0.475	0.536	0.635	0.667	0.643					
			3	0.532	0.530	0.494	0.530	0.406	0.686	0.630					
			4	0.544	0.541	0.620	0.541	0.718	0.674	0.639					
			5	0.530	0.527	0.485	0.527	0.912	0.698	0.619					