DeepConvLSTM - Forecasting Volatility of Stock Index Using Gramian Angular Field and Convolutional LSTM

dydx201822139, azure-fog-201706586, BusyApe201804930

Group 17

Abstract

The task to predict the movement of the stock market is publicly known to be a challenging one. The stock market's inherit uncertainty and randomness justifies it, but also compounds the difficulty of it. Besides, the nature of the stock market being active only for a fixed duration during the day hinders the forecasting power of models. In this paper, we propose the *DeepConvLSTM* model that is designed around this oberservation. By utilising the image conversion techniques of Gramian Angular Field (GAF), we partition the stock market index data by trading days. The rationale for this is to better extract intra-day and inter-day patterns, when using Convolutional Long Short-Term Memory (ConvLSTM) and Convolutional Neural Network (CNN) based architecture. The aim of this is to forecast the absolute within day return, as a representation of volatility, of the upcoming day. Contributions to this problem can help investors gauge the level of risk that they face in the upcoming trading day and assist in strategising for the most effective trading plan. Based on the US500 and UK100 indexes, the results in paper suggests that the *DeepConvLSTM* performed better than benchmark models. It yields superior forecasting ability, compared to traditional forecasting models like the Autoregressive Integrated Moving Average (ARIMA), and novel models that were similarly based off of image conversion techniques.

Contents

1 Introduction

The stock market is a setting where numerous trading and investments take place. One of the most important piece of information within the stock market is the market index. It is a composite index of the prices of the many stocks that are traded in it, and thus gives a broad reflection of the general market sentiment and macroeconomic environment. Due to the above, forecasting for the market index is a task with substantial implications. For instance, accurately predicting the future movement of the index may dictate how investors plan their investment strategies and yield massive financial consequences. Yet, research has shown that volatility and unpredictability are key characteristics of the stock market (Bhowmik and Wang, 2020), and it is ever more prominent post-2008. This makes it especially challenging to do forecasting.

Time series forecasting has been the focus of research for a very long time (Aliev et al., 2004). In recent years, with the fast development of machine learning, more advanced tools are used in this area. Chatterjee et al. (2000) tries to systematically use neural network to make accurate forecast of stock market movement. As deep learning being studied extensively (Goodfellow et al., 2016), researchers tried to apply different models that are suitable for time series forecast. Silva et al. (2021) use generative adversarial nets to quantity uncertainty to predict the spread of Covid-19. Desai et al. (2021) applies another generative model called VAE and invents TimeVAE for multivariate time series generation. Bloemheuvel et al. (2022) take the benefit of graph neural network to process long sequence in multivariate time series regression task. Also, people have used basic multilayer perceptron, RNN, CNN and LSMT to do different forecasting tasks (Schmidhuber, 2015). For example, Chen et al. (2016) focus on improving current results by planar feature representation methods and deep convolutional neural networks.

Among the above approach, convolutional neural network (CNN) is one of the most popular neural network structures and has been used a lot in different tasks (Dhillon and Verma, 2020). Ker et al. (2017) review the application of CNN in medical image analysis as CNN is efficient to do detect patterns. Yang et al. (2019) apply CNN to improve the result of single image super-resolution and overcomes previous limitations. Besides, it is widely used in facial recognition, which aims to identify a person based on certain aspects of his physiology and has recently become the dominant approach in this area (Hassan, Abdulazeez, et al., 2021). In economics, Tuo et al. (2021) design a CNN based model to predict economics development of industries of different regions.

More recently, researchers start to use various deep learning tools to convert time series into graphs. The visualization helps to do classification and improve forecasting results. Hermansky (1990) uses perceptual linear predictive technique to improve speech recognition system. Donner et al. (2011) analyse time series based complex networks especially recurrence plots. Wang and Oates (2015) propose a novel framework which combines Gramian Angular Summation/Difference Fields and tiled convolutional neural network. Chen and Shi (2019) use Relative Position Matrix (RPM) and Convolutional Neural Network to convert time series data to 2D images and improve the accuracy of time series classification.

Related to stock forecasting specifically, the temporal structure of stock data motivated many sequence-based models to be the core to many past solutions. For instance, structures such as the Long Short-Term Memory (LSTM) were previously utilized to predict US stock prices (Moghar and Hamiche, 2020). Liu et. al. (2019) used it to forecast volatility in the stock market and outperformed tradtional methods like Support Vector Machines. Also, techniques in computer vision such as CNN were adopt in some proposed models too. For example, researchers implemented a 1-dimensional CNN to analyse financial time series and make forecasts for it (Rasheed et al., 2020). Some models such as the CNN-LSTM model even combined the types of model structures above to forecast Chinese stock prices and showed great success compared to other forecasting options (Lu et al., 2020).

In this paper, we will not be focusing on as broadly as predicting the movement of the stock price over multiple periods. Instead, we aim to do time series prediction on the daily volatility of the stock index next day, based on the history of the stock index of a certain length in time. Although the scope is narrower, it is not a trivial task either – having some expectation of the amplitude of the ups and downs in the stock market for the next day is still helpful for investors to optimise their portfolios, especially the ones who engage in short-term trading. It captures the degree of fluctuation that may occur the next day and the gains that may potentially be realised when trading and exploiting those fluctuations on that day. On a broader context, volatility is important information when financial options are priced. Also, macro policymakers often view market volatility as an key indicator when determining their policy response (Poon and Granger, 2003).

We propose a model that is based on processing time series as a form of images. This is done by image conversion with Gramian Angular Field (GAF). The core of the neural network will be based on multiple layers of convolutional long short-term memory (ConvLSTM). Therefore, it will be known as *DeepConvLSTM*. It is inspired by the work of Hong et. al. (2020), where time series was forecasted using GAF and ConvLSTM networks based models. Note that throughout this article, we will interchangeably use "price" as an substitute term for "market index".

The structure of this paper is as follows. We will first defining the core problem that we wish to tackle and the data that we will be using for it. Then, we proceed to formulate our proposed model. Following it, we will discuss the results from our model. We will then evaluate our model and compare against other benchmark models. Lastly, we will conclude and discuss future directions.

1.1 Aim

Let us begin by formally defining the forecasting problem that we are focusing on. Let $\{p_i\}_{i=1}^{T}$ be the time series that we are interested in. In this case, it will be the entire recorded market index. The fundamental problem of time series forecasting is to determine the optimal model that gives the most accurate prediction to the time series next period, given the history of it. What we try to achieve here is to predict the volatility of the market index of next day. Instead of using standard deviation to represent volatility, we will consider an alternative measure of it. Many research have suggested that absolute returns to be a good proxy for volatility (Ding et al., 1993). Thus, we will model volatility using absolute within-day returns and treat it as our forecasting target. We define absolute within-day return as the magnitude of the change in the log of the first price of the day to the log of the last price of the day. In other words, the absolute within-day return y_t of day t is defined as

$$y_t := |\log(p_l) - \log(p_s)| \tag{1}$$

where p_s and p_l are the first and last prices recorded on day t, respectively. Consequently, on day t we want to use the history of the market index $X_t = \{p_i\}_{i=t}^{t+\tau}$, to predict the absolute within-day return of next day y_{t+1}. To be more precise, we want to determine the model f such that y_t is predicted by $\hat{y}_{t+1} := f(X_t)$. In other words, it is a sequence-to-one prediction problem. If we put it in the context of deep learning modeling, given some loss function L, we aim to solve:

$$f \in \arg\min_{g} \; L(g(X_t), y_{t+1}) \tag{2}$$

However, it is crucial that we recognise market index having a different characteristic as to some other financial data. That is, the data has gaps. For instance, currencies are traded 24/7. Consequently, data for exchange rates is recorded as time series over such periods too. On the other hand, stock markets are only open for a certain period every day. Even though stocks may still be traded outside of market hours, there are still periods of the day when stocks cannot be traded. Depending on the context and the nature of the data, having gaps in the time series may not always be problematic. The issue here, however, is that the flow of information does not halt as trading does. For example, due to time zone differences, news about a country on the other side of the globe may well impact investors' sentiment towards the macroeconomic environment of the local market. In other words, the intrinsic prices or value of stocks may change, whilst the recorded prices remain static. Theoretically, this is a direct application of the Efficient Market Hypothesis, where prices within the stock market is very efficient in incorporating any new information (Malkiel, 2003). Naturally, this will have an impact on market indices. Therefore, it justifies the need to view the daily data as disjoint entities.

With this observation in mind, the forecasting problem that we want to tackle here is adjusted slightly. Instead of viewing the history X_t as a typical sequence of past market index values, it should be considered as a sequence of daily market index sequences. In other words, $X_t = \{\{p_{i,j}\}_{i=1}^{T_D}\}_{j=t}^{t+D}$, where D is the number of days we consider within the sequence.

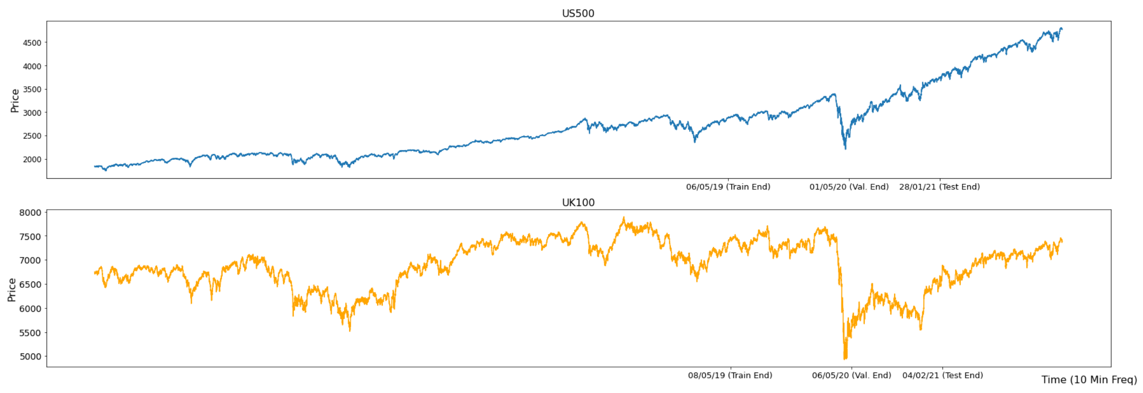

Figure 1: US500 & UK100 Time Series Data

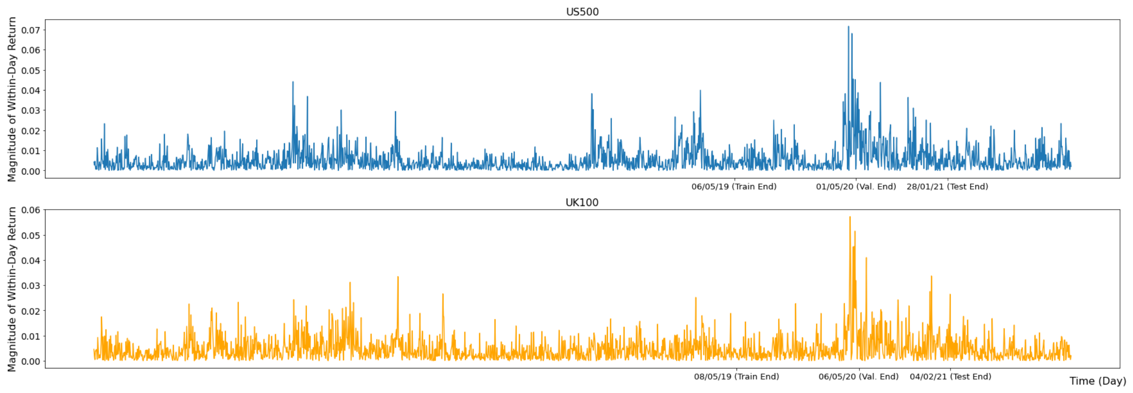

Figure 2: US500 & UK100 Magnitude of Within-Day Returns

1.2 Dataset

The main source for our data on market index is Dukascopy Bank Historical Data Feed. Stock market index CFDs are used as the trading time is easy to trace with gaps between days. More specifically, this paper uses data from January 1, 2014 to December 31, 2021 to study the most recent nine years' price. The data are chosen by excluding the non-trading hours and weekends to separate intraday price movement. Note that due to reasons during implementation, which will be discussed later, the data used in reality will not be this long in length. Also, 10-minute frequency is applied to avoid heavy data load but at the same time gives reasonable sample size to learn GAF image. The market indexes that we will consider are the US500 and the UK100. The former tracks the 500 large companies that are listed in the US stock exchanges, whilst the latter tracks the 100 largest companies listed in the London Stock Exchange. But due to constraints in time and computation power, our focus will be primarily on the US500 index.

The data in its raw form records every 10 minutes, the following 5 values for each bid price and ask price: open price, close price, high price, low price, volume. This paper obtains a single bid price b_t and ask a_t, by taking the mean of their respective open, close, high, and low prices. A single final price is then determined by taking the weighted average of bid and ask price using the bid volume v_{bt} and ask volume v_{at}. In other words, the final price p_t is found by

$$p_t := \frac{v_{at}b_t + v_{bt}a_t}{v_{at} + v_{bt}} \tag{3}$$

This weighted average is able to adjust the price based on whether the buy-side or the sell-side of the stocks is more dominant. Opening trading hours are identified by removing the days where trading volume is zero. Finally bid and ask data are merged to get the complete dataset. The plots of the full original time series data are presented in Fig. 2. In addition, the data for the output, i.e. the daily volatility of the stock index, is displayed in Fig. 2.

2 Modelling

2.1 Preliminaries

2.1.1 Gramian Angular Field

Unlike creating a model that inputs the time series directly, we first convert the time series into images. Specifically, we separately convert the data into an image form known as Gramian Angular Field (Wang and Oates, 2015). Given any 1-dimensional time series $\{p_i\}_{i=1}^{T}$, the process to generate its GAF image representation is as follows: We apply min-max normalization on the time series, such that all the values of the data are between 0 and 1. Given the re-scaled time series $\{\tilde{p_i}\}_{i=1}^{T}$, we determine its polar coordinate and determining its angular position $\{\phi_i\}_{i=1}^{T}$, where $\phi_i = \arccos(\tilde{p_i})$. In the end, for each $i, j \in \{1, ..., T\}$ pixel of the GAF image x_{ij}, it will be found by $x_{ij} = \cos(\phi_i + \phi_j)$. This particular procedure that we have described is to generate the Gramian Angular Summation Field, which is a particular class of GAF. It is noteworthy that the choice to scale the data into the interval between 0 and 1 is significant. If we know the minimum and the maximum values of the original time series, it is possible to invert the GAF back to the original time series. This property is useful for us to establish that the GAF form of a time series is indeed an equivalent representation of the original time series itself. GAF is able to preserve the temporal correlations and dependencies of its original time series. The application of Gramian Angular Field in Deep Learning was first proposed to work with classification and imputation problems (Wang and Oates, 2015).

2.1.2 Convolutional Long Short-Term Memory

Convolutional LSTM (ConvLSTM) is a neural network structure first proposed to work on problems that involved processing spatiotemporal data. In the original paper, maps of rainfall were processed to predict future precipitation levels (Shi et al., 2015). In order to understand how ConvLSTM works, we must first discuss the mechanism behind LSTM and covolutions with images.

LSTM were initially proposed to deal with the inability of Recurrent Neural Networks (RNN) to learn long-term dependencies (Hochreiter and Schmidhuber, 1997). Similar to RNN, it has a structure where data $\{x_j\}_{j=1}^{T}$ and states of the network $\{h_j\}_{j=1}^{T}$ are sequentially connected with many cells. In RNN, every cell t outputs a new state h_t by considering only a single neural network layer. That is, it applies some activation function on the matrix multiplication between the previous state h_{t-1} and the new input x_t.

On the other hand, apart from working with the state h_t and data x_t, each LSTM cell t additionally outputs the cell state C_t, and inputs the previous cell state C_{t-1}. Instead of describing the original LSTM, we will describe the "Peephole" variant of LSTM, as it is the most related to ConvLSTM (Gers and Schmidhuber, 2000). Firstly, given the new input x_t, the LSTM will want to consider what information from the previous state h_{t-1} to forget, condition on the previous cell state C_{t-1}. This is done by the forget gate f_t, i.e.

$$f_t = \sigma(W_{xf} \cdot x_t + W_{hf} \cdot h_{t-1} + W_{Cf} \cdot C_{t-1} + b_f) \tag{4}$$

where σ is some activation function, typically the Sigmoid Function, and $\cdot$ is the matrix multiplication operator. Secondly, the cell will determine what values to keep to use to update the cell state. This is done by the input gate i_t, i.e.

$$i_t = \sigma(W_{xi} \cdot x_t + U_{hi} \cdot h_{t-1} + W_{Ci} \cdot C_{t-1} + b_i) \tag{5}$$

Thirdly, the cell will want to determine what information from h_{t-1} and x_t that we want to use for outputting the new state h_t. This is done by the output gate o_t, i.e.

$$o_t = \sigma(W_{xo} \cdot x_t + U_{ho} \cdot h_{t-1} + W_{Co} \cdot C_t + b_o) \tag{6}$$

Now, to incorporate the new object, being the cell states, into the structure of the LSTM cell, we need to consider how it is updated. A 'candidate' for the new cell state $\tilde{C_t}$ will be generated by

$$\tilde{C_t} = g(W_{xC} \cdot x_t + W_{hC} \cdot h_{t-1} + b_C) \tag{7}$$

where g is some function. It common that $g(x) = \tanh(x)$. Using it, alongside the forget gate f_t, input gate i_t and the previous cell state C_{t-1}, the new cell state will be determined by

$$C_t = f_t * C_{t-1} + i_t * \tilde{C_t} \tag{8}$$

where $*$ is the element-wise product operator, also known as the Hadamard product. Lastly, we want to create output the state h_t using the output gate o_t, while adjusting it based on the new cell state C_t. This is done by the following calculation:

$$h_t = o_t * g(C_t) \tag{9}$$

Note that the sets of parameters,

$$W := \{W_{xf}, W_{xi}, W_{xo}, W_{xC}, W_{hf}, W_{hi}, W_{ho}, W_{hC}, W_{Cf}, W_{Ci}, W_{Co}\}$$

$$V := \{b_f, b_i, b_o, b_c\}$$

contain all parameters that characterise an LSTM structure, and that they are the same for every cell. That being said, they the parameters that we train when optimising any overarching neural network.

Regarding convolutions, we will only discuss about 2-dimensional convolutions specifically. The concept can be easily generalised to convolutions of different dimensions by slightly altering it. In essence, convolution on some object requires a kernel to produce a feature map. In the simplest case, let us consider the case with 2-dimensional piece of data $X = \{x_{ij}\}$, where $i = 1, ..., n_h$ and $j = 1, ..., n_w$, and there is only a single channel. In other words , X has n_h rows and n_w columns. Now let $\bigodot$ be the convolution operator and that the kernel with k_h rows and k_w columns be defined to be $\beta := \{\beta_{ij}\}$, where $i = 1, ..., k_h$ and $j = 1, ..., k_w$. When we evaluate $\alpha := \{\alpha_{ij}\} = \beta \bigodot X$, we yield that for all $i = 1, ..., n_h - k_h + 1$ and $j = 1, ..., n_w - k_w + 1$,

$$\alpha_{ij} = \sum_{p}^{k_w} \sum_{q}^{k_h} \beta_{pq} x_{i+p-1, j+q-1}$$

Here, the resulting output will be a 2-dimensional object with $n_h - k_h + 1$ rows and $n_w - k_w + 1$ columns. The operation interacts the same kernel with all the different parts of the input object, i.e. the parameters are shared. This gives it property of translation invariance, especially when working with images. In the context of deep learning and convolutional neural networks, the kernel β contains parameters that are trainable.

What we have calculated above is assuming that both the stride of the convolution operator and the number of filter for the kernel are 1. Should we wish to generalise this to cases where the number of strides is larger, the output feature map will have a slightly smaller size. With more number of filters in the output feature map, a different kernel will be used to project each filter. Moreover, when the number of channels of the input data is greater than 1, there will be a different kernel for each channel too.

Now with the concepts of LSTM and convolution operator in mind, we can properly define what ConvLSTM does. ConvLSTM has the same structure as LSTM does with a sequence of cells. Unlike LSTM, the interaction of the objects at each gate are not matrix multiplication, but convolutions. In other words, within each ConvLSTM cell, the matrix multiplication operator $*$ is replaced by the onvolution operator $\bigodot$ in equations (4)-(9). This also implies that the set of trainable weights W in LSTM, would now instead be containing kernels which we will be training in ConvLSTM. Effectively, when the 2-dimensional objects (cell states, states, inputs) interact with each other at the gates, their spatial features and relations have significance within the cell. Therefore, ConvLSTM allows spatial features of images to be recognised when they are processed sequentially over the cells at different time periods.

2.1.3 Loss Functions

The loss function guides the optimisation process of our model. As shown in 2, the loss function L that we choose, is part of how we determine our model f. For regression problems, Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE) are typical choices for the loss function. Given targets $\{y_i\}_{i=1}^T$, inputs $\{X_i\}_{i=1}^T$, model f, and predictions $\{\hat{y}_i\}_{i=1}^T$, where $\hat{y}_t = f(X_t)$, they are defined as:

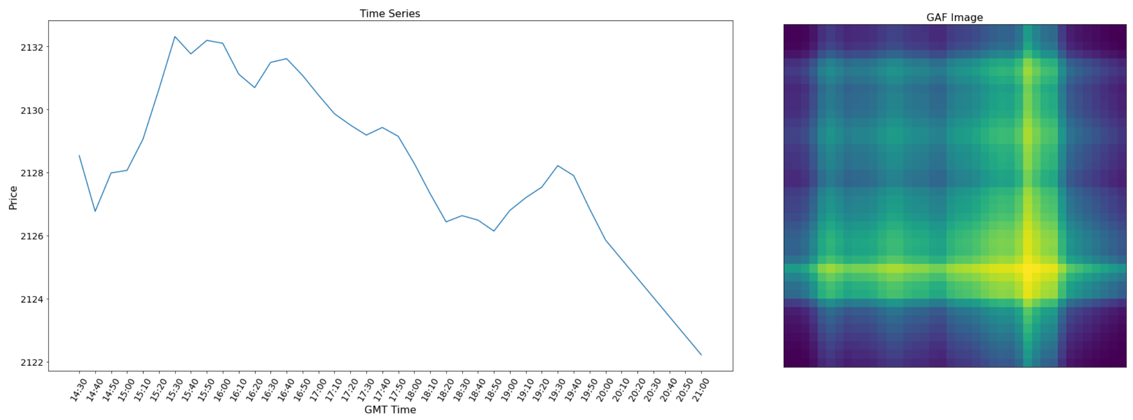

Figure 3: US500 Time Series on May 6th, 2019 & its GAF Representation

$$RMSE := \sqrt{\frac{1}{T}\sum_{i=1}^{T}(\hat{y}_i - y_i)^2}$$

$$MAE := \frac{1}{T}\sum_{i=1}^{T}|\hat{y}_i - y_i|$$

We should note that when we optimise the model by trying to minimise the RMSE, due to it being a monotonic transformation to Mean Squared Error (MSE), it is equivalent to just minimising MSE. MSE is defined as:

$$MSE := RMSE^2$$

Nonetheless, RMSE is a better metric to examine, as it neutralises the effect of the quadratic exponent. This is especially useful when we seek to compare RMSE and MAE, where they would be expressed in the same unit.

2.2 Preprocessing

With our market index data being unified into a single price using (3), we have to process the data into a suitable form. For the input of the model, we will want to convert the data into GAF images. Given our forecasting problem based on the market index and the nature of the time series, it is only reasonable to apply the image conversion for the data within each day. Therefore, we will convert each of the daily time series into a GAF image. Fig. 3 shows is an example of a daily time series of the US500 on May 6th, 2019, and its corresponding GAF image.

Moreover, since we are trying to use the history of the market index to predict about the information about the upcoming period, we want to form consecutive subsequences using the GAF images. Since the number of trading days within a calendar month is approximately 20, we will choose the length or timestep of the subsequences to be $D = 20$. From this process, the input data will be a 4-dimensional object (or tensor) with dimensions $T \times D \times T_D \times T_D \times 1$, where T_D the number of prices recorded within each day, and T being the total number of days. In terms of the output, we simply calculate the volatility of the market index for each day, as defined in (1). This would result in it being a 1-dimensional tensor, with dimensions $T \times 1$.

The value for T_D may differ for different stock indices. In our case, the US500 index has $T_D = 40$, whilst $T_D = 52$ for the UK100 index. This is due to the varying lengths in trading hours between stock exchanges. The value for T is not as straightforward as extracting the total number of available days in the data minus the length of the input subsequences. In order for us to fully utilise the capabilities of ConvLSTM, or more simply LSTM, in having memory in long-term information, during implementation, we want to set it to be 'stateful'. By having this setting, it imposes the constraint on the data to be divided into strictly full batches. In other words, the total number of samples in the data we use will have to be divisible by the batch size B. Cnnsequently, part of the data cannot be used. In this case, we chose to remove the most recent data points. Including the days without trading, the resulting data for the US500 time series lasts from January 1st, 2014 to January 28th, 2021. For the UK100 time series, it lasts from January 1st, 2014 to January 1st, 2014 to February 4th, 2021. After omitting the days with no trading, both the US500 and UK100 data have $T = 1792$ total number of days.

For any Deep Learning model, it is paramount that the data is split into training, validation, and testing sets. As mentioned already, however, when data is passed used in the model, its size must be divisible by B. This fact remains relevant for each of the three parts of the data. Therefore, the data was segmented on basis on this divisibility aspect in mind. The consequence of this was that for both datasets, the first $T_{train} = 1344$ days are for training, the following $T_{valid} = 256$ days are for validation, and the next $T_{test} = 192$ days are for testing. The specific dates are displayed in Fig. 1.

Lastly, data are normalised within its own series, using max-min scaling in order to maximise model training efficiency. Although GAF images are used as inputs as well, but since they are projected into the interval between 0 and 1, further normalisation is unnecessary.

2.3 Architecture

Indeed, the application of GAF and ConvLSTM in time series was already attempted by Hong et. al. (2020). In their work, they aimed to forecast solar irradation for the next timeframe. The model that they proposed was an encoder-decoder netowrk which incorporated ConvLSTM and passed GAF images as data to obtain the GAF for next period. After optimising for the encoder-decoder network, the output GAF image was later inverted to retrieve the time series of the target. The encoder-decoder network consisted of a single ConvLSTM layer, followed by a 3-dimensional convolution layer, and batch normalisation after every layer.

Inspired by their model, the *DeepConvLSTM* model that we seek to bring forth has an architecture that uses ConvLSTM and convolution layers as the bedrock. We hope to use such a sequential image learning network to uncover underlying spatiotemporal patterns within the relative fluctuations and dependencies that GAF images contain about the time series, which then would enhance the forecasting power of the model. Rather than having only a single set of ConvLSTM and convolution layer, however, our model extends the idea and includes multiple multiple blocks of ConvLSTM and CNN (ConvLSTM-CNN block). This idea to extend the depth of the network is based on the work done by Eldan and Shamir (2016), where it was shown that for a forward feeding network, increasing the depth of the network by one level already yields result that improves exponentially. Also, our model does not require GAF inversion, since we are aiming to forecasting a single value only, that is the volatility of the next day.

2.3.1 ConvLSTM-CNN Block

Within each block, the layers are in the order of ConvLSTM, batch normalisation, convolution, activation, convolution, activation, convolution, batch normalisation, activation. The rationale for the structure of each block is to combine ConvLSTM and usual convolutions is to track long and short terms patterns more effectively. ConvLSTM incorporates the spatial features when trying to identify long term patterns through LSTM. On the other hand, the convolutions that follow helps extract finer spatial features within the objects. Put into context, ConvLSTM can focus on the inter-day patterns whilst the convolutions can manage the intra-day patterns. This is supported by the work done by Hong et. al, where the ordering of ConvLSTM followed by a CNN yielded decent performance.

Moreover, in order to have stronger pattern recognition power in within the model, we employ a CNN structure that was used in the *Network in Network* (NiN) neural network, which saw great success in image recognition (Lin et al., 2013). Namely, the final 2 convolution layers within each block has a kernel of size 1 by 1 only. This is to improve the model's power to recognise patterns from the location of each pixel. The kernel size of the ConvLSTM and the first convolution layer will be the same for each block. We will specify this hyperparameter in the forthcoming section.

The choice of activation here is the Rectified Linear Unit (ReLU), as it has been a common selection for various modern CNN. It enables better training by avoiding vanishing gradients and also it is computationally simpler than some other alternatives such as the sigmoid function. It is worth noting that after most of the layers, we will apply batch normalisation. This is to help the model to be more stable during training, and to potentially speed the process up. However, after the final convolution layer, the batch normalisation is applied before the ReLU activation. This is justified in the original paper on batch normalisation, which suggested that applying batch normalisation prior to applying non-linearity is the better option than the opposite (Ioffe and Szegedy, 2015).

Regarding the ConvLSTM layer, it has already been noted that it will be set to be stateful. In addition, choice of activation within the ConvLSTM cells will be the hyperbolic tangent function and the choice of the reccurent activation will be the sigmoid function. Both of which are common choices for models that utilise some form of LSTM.

Lastly, since we are working with convolutions here, the number of strides is a necessary hyperparameter to be chosen. To be safe and to maximise the power of the model, we set the stride of all of the convolutions (including ConvLSTM) to be 1. This is such that information on every position of on the image can be processed and used to train the model. Another relevant hyperparameter is the number of filters used within each block. It will be kept the same for the ConvLSTM and convolution layers. The value of this hyperparameter will be specified in the subsequent section.

2.3.2 Overall Neural Network

With the ConvLSTM-CNN block described, the overall network can be now be defined. The input data is passed through 3 consecutive sequence of ConvLSTM block followed by some form of pooling layer. Specifically, the first two blocks will be followed by a max pooling layer, whilst a global average pooling layer will come after the final block. A dropout layer follows to regularise the model. Finally, another fully connected layer is appended at the end, corresponding to the output. Since we the output we have here is a one-dimensional object, the output fully connected layer only has one neuron. Whenever a layer has the option to include padding, we will do so to enable the network to be more flexible with the shapes of input data. A diagram of the overall network for US500 data is presented in Fig. 4. The analogous diagram for the *DeepConvLSTM* model that corresponds to UK100 data can be made by changing the width and height dimensions of the input images from 40 to 52 in Fig. 4, and following the operations of each layer.

We want to comment to the usage of the pooling layers in our model. Pooling layers can reduce the size of the data object and summarise the local neighboring parts of the data for lower level patterns. This typically enables images, for instance, to be positional invariant and would be computationally faster to process in the coming layers. Max pooling identifies the sharpest parts of the data, which is useful for our cause where financial data has many steep peaks and troughs. However, we do not want to be overly extreme and forgo the cyclic nature of the stock index, and there is often a trend that the index gravitates back to. Therefore, global average pooling is applied in the end instead. Besides, global average pooling serves as a good way to flatten the data without imposing excessive number of parameters to the model. This is a similar approach to the ResNet architecture, compresses the 2-dimensional object in the end using global average pooling (He et al., 2015). The architecture was highly successfully at recognising patterns within images.

Next, it is necessary to specify the number of filters for the ConvLSTM and convolutions for each of the ConvLSTM-CNN block. In the order of increasing network depth of the block's location, the filter sizes are 4, 16, 64. The increasing filter size is selected such that we may extract more complex patterns in the data, the deeper it is within the network. However, such filters sizes is comparatively smaller to more modern networks. This due to the constraint of limited computation power, and to avoid long training times.

As noted already, the kernel sizes for the ConvLSTM and the first convolution layer within each of the blocks are equal. The kernel size associated to the first ConvLSTM-CNN block will be 5 by 5. For the rest of the blocks, it will be set to 3 by 3 instead. The justification for this is that the model would be able to extract more spatial information from the initial input data block.

The ConvLSTM inside the first two ConvLSTM blocks returns the states for the entire sequence. This is to preserve the sequential nature of the data, and that the object and be further processed in another ConvLSTM structure. Due to the aim of our model, which is to predict about the next day only, the final ConvLSTM only returns the latest state. As a consequence, the convolution layers in the final block are 2-dimensional convolutions, and in the two blocks they are 3-dimensional convolutions, where the third dimension of the images is the timeframe. To supplement the above, we include max pooling after each block, where the pooling window is 2 for each dimension. In a typical

CNN, max pooling after each block serves the purpose to aggregate the object, in hopes to identify lower level patterns. Specifically, for the first two ConvLSTM blocks, we apply 3-dimensional max pooling, which further does aggregation over the timeframe. This implementation aims to identify trends over the past history, and allows the subsequent ConvLSTM-CNN block to work on lower level trends.

2.4 Implementation

The models will be trained using the `TensorFlow` library (Martín Abadi et al., 2015), alongside the `Keras` API (Chollet et al., 2015). The conversion of time series into GAF images will be done using the `pyts` library (Faouzi and Janati, 2020).

During training, the choice for the loss function and optimisation method is important. We will be training our model using MSE as loss and the Adam optimisation algorithm. All of the settings for the Adam optimisation will remain as the default setting in `Keras`, except the learning rate. The learning rate will vary depending on the model that we train. Also, we do not wish to shuffle the data when training nor when spliting the data set. This is to preserve the temporal sequence that the time series already has, and that the fundemental reason why time series forecasting is tricky is due to this natural sequential aspect of it.

Regarding the choice of the batch size, it has been noted in literature that a setting a higher batch size will typically worsen the ability of a model to generalise (Keskar et al., 2016). Conversely, setting a smaller batch size will lengthen the training duration, as more steps are needed for during each epoch. Therefore, we select a moderately-sized batch size when training the models. That is, we set $B = 64$.

3 Results

The main results of our experiment and model are presented in Table 1. When trained and tested using the US500 data, the *DeepConvLSTM* model that we propose obtained a test RMSE of 0.1102 and a test MAE of 0.0724. Such values are similar when the model was trained and tested with the UK100 data too, where RMSE and MAE are 0.1088 and 0.0756, respectively. Since the model performed similarly on both data sets, it offers evidence that our model has some ability to generalise to other stock index.

Data	***DeepConvLSTM***		**ConvLSTM**		**ARIMA**		**1D CNN-LSTM**		***DeepConvLSTM* + Max & Min**	
	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE
US500	0.1102	0.0724	0.1196	0.0802	0.1129	0.0862	0.1169	0.0993	0.1186	0.0775
UK100	0.1088	0.0756	0.1198	0.0840	0.1177	0.0878	0.1241	0.1084	0.1026	0.0776

Table 1: Performance of Models on Test Data

3.1 Model Diagnostics

It is critical that we examine the training process for the model itself, in order to gauge whether or not there are inherent problems to the model or the data. Fig. 5 and 6 show the plots of the losses over each training epoch corresponding to training on US500 and UK100 data, respectively. We it is evident that for the training losses, it converges how it should ideally converge, which is a good sign. The possible source of concern is that, despite already imposing regularisation from dropout, the validation loss decreases very little. Should there be no regularisation, it is likely that the validation loss will be significantly higher than the training loss. This may well be a sign of bias and that the validation data is distributed differently than the training data is. Therefore, it is very difficult to recover strong patterns from that training data that also are fully applicable to the validation data. Nonetheless, the inclusion of the dropout is able to keep the validation loss close to training loss, which is positive for our model's performance at generalising to unseen data.

Even although validation loss did not drop much, if we inspect it alongside test losses, we find that the losses from evaluating the model on test data are in fact lower than the validation loss by some

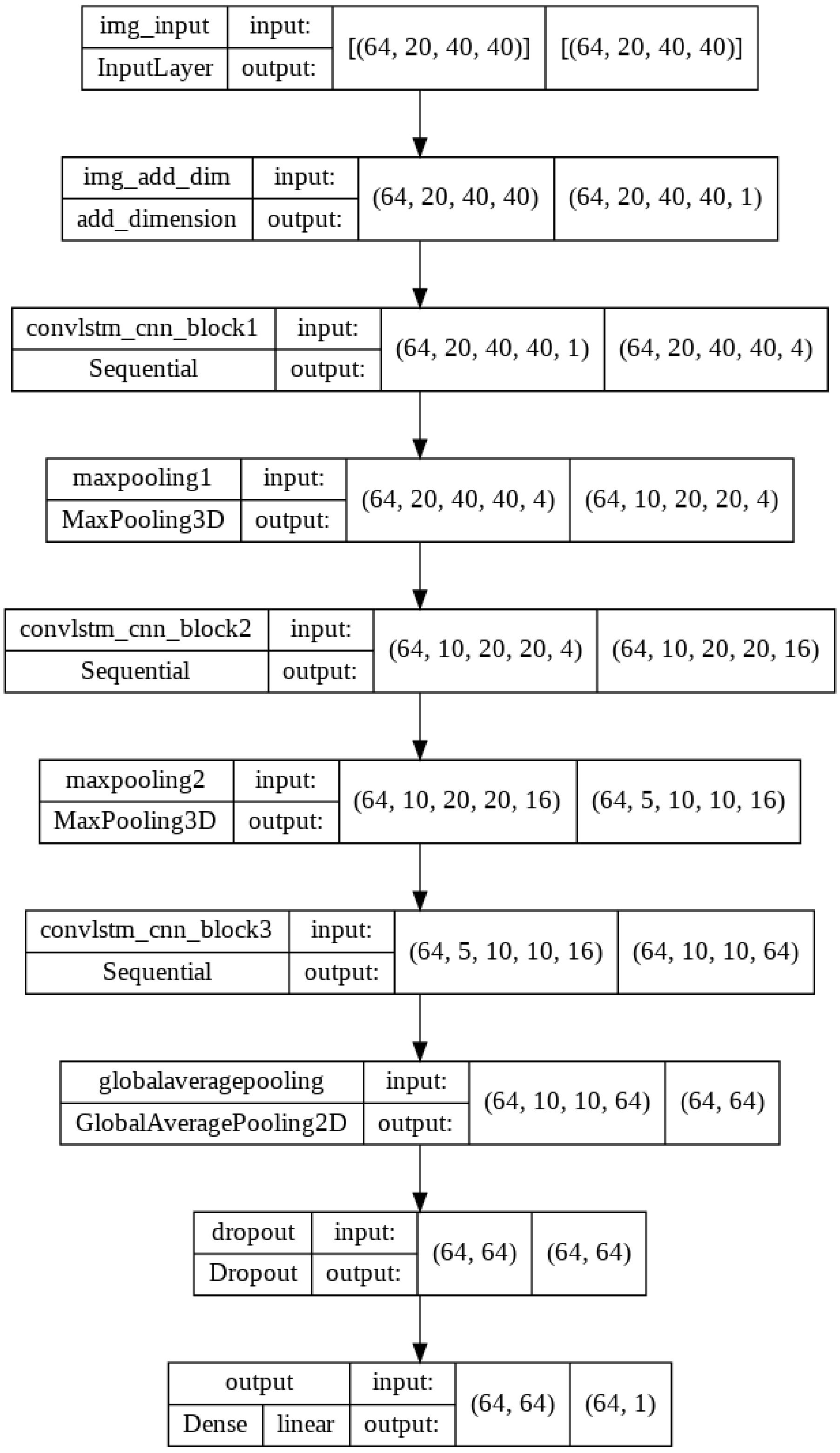
img_input | InputLayer | input: | output: | [(64, 20, 40, 40)] | [(64, 20, 40, 40)]
img_add_dim | add_dimension | input: | output: | (64, 20, 40, 40) | (64, 20, 40, 40, 1)
convlstm_cnn_block1 | Sequential | input: | output: | (64, 20, 40, 40, 1) | (64, 20, 40, 40, 4)
maxpooling1 | MaxPooling3D | input: | output: | (64, 20, 40, 40, 4) | (64, 10, 20, 20, 4)
convlstm_cnn_block2 | Sequential | input: | output: | (64, 10, 20, 20, 4) | (64, 10, 20, 20, 16)
maxpooling2 | MaxPooling3D | input: | output: | (64, 10, 20, 20, 16) | (64, 5, 10, 10, 16)
convlstm_cnn_block3 | Sequential | input: | output: | (64, 5, 10, 10, 16) | (64, 10, 10, 64)
globalaveragepooling | GlobalAveragePooling2D | input: | output: | (64, 10, 10, 64) | (64, 64)
dropout | Dropout | input: | output: | (64, 64) | (64, 64)
output | Dense | linear | input: | output: | (64, 64) | (64, 1)

Figure 4: *DeepConvLSTM* Overall Architecture

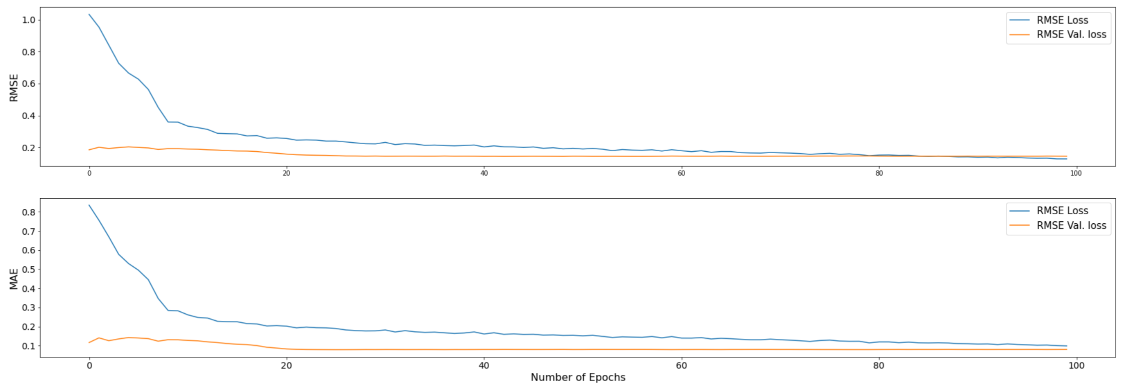

Figure 5: *DeepConvLSTM* Training and Validation Loss on US500

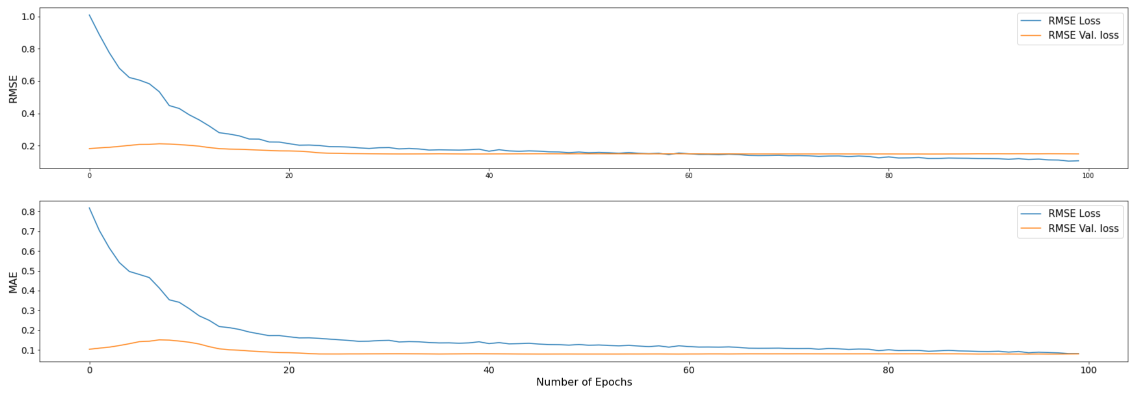

Figure 6: *DeepConvLSTM* Training and Validation Loss on UK500

margin. XXX This draws our attention that possibly the validation data is significantly more complex or distinct from the other segments of the data. Examining Fig. 1 and 2, we can immediately identify that the validation data contained a huge shock in the market index and spike in absolute within-day returns. Given the timing, it is highly probable that this was due to the global coronavirus pandemic. Such an event can be classified as a massive outlying event, and that the behaviour of the market index clearly differed from was it was prior to it. Therefore, it is not a surprise that training the model using data in such an order is incapable of lower validation loss by much. In addition, the same two figures show indicate how after the shock in to the time series, the stock index was restored to movement that was more regular. This coincides with when the test data is selected, and would explain why the model has greater forecasting power on test data instead.

4 Evaluation

4.1 Model Comparison

It is meaningless just by considering the model losses in absolute terms. Let us consider some benchmark models to compare our model with. We consider 3 candidate models: ConvLSTM, 1D CNN-LSTM, and ARIMA.

ConvLSTM model is the model that is based around the work by Hong et. al (20190, where it contains only a single convolutional LSTM layer and convolution layer afterwards. We aim to replicate the architecture as much as possible in terms of the hyperparameters, but will adjust it in order to adapt to solving our absolute daily returns forecasting problem. For instance, our variation of ConvLSTM will still choose kernel size of 3 by 3 for the ConvLSTM layer and 5 by 5 for the convolution layer, same as the original authors did. However, we will adapt it by only having a single unit dense layer in the end, since our problem's output is single-dimensional, whereas theirs were multidimensional.

The second model that we will compare against is the 1D CNN-LSTM model. Instead of inputting a

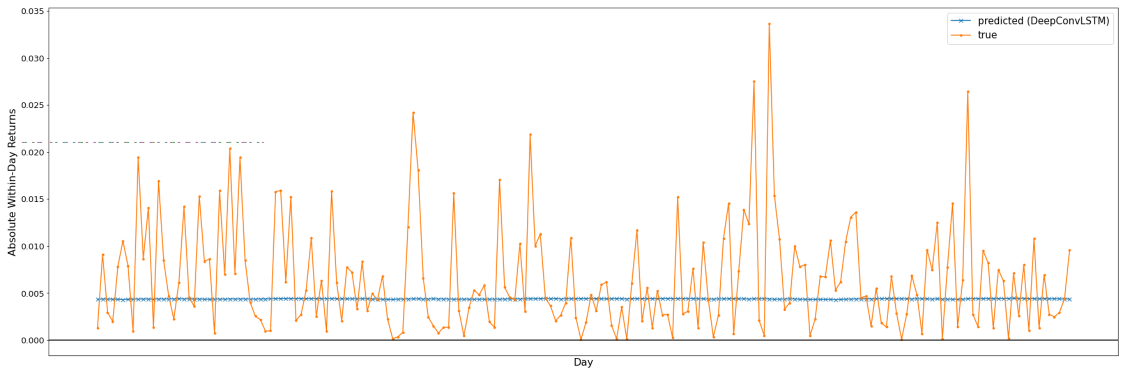

Figure 7: Predicted Values by *DeepConvLSTM* on US500 Data

subsequence of images, it accepts a subsequence of the daily time series directly. For each daily time series within this subsequence, it is processed through a 1-dimensional CNN to extract key temporal features. Such model does not rely on image conversion technique, and the 1-dimensional CNN sees the temporal dimension of the time series as the spatial dimension to try to find low-level intra-day patterns over time. Multiple 1-dimensional convolution layers will be used and max pooling applied in between. The output of the CNN part of the model is then analysed by LSTM to further analyse the other temporal dimension. That is, it would try to seek for inter-day time dependencies within the subsequence it takes as input. Finally, the output is generated after the LSTM.

The last model is the autoregressive integrated moving average (ARIMA) model. It is a model which is founded more on classical statistics-based concepts and techniques. It is based off of the belief that the sequential aspect of time series lead to residual of past values having an effect on new values. Therefore, it serves well to be a decent benchmark for how our deep learning based approach may compare against more tranditional methods. Note that 3 parameters are needed to fully characterise an ARIMA model. They are the lag order, degree of differencing, and the order of moving average. For the sake of simplicity, we will simply use functions that automatically determine the optimal choices for those three parameters.

After training the 3 new models, the results are appended to Table 1. From the table, we have underlined the model that minimises loss in each category. We can see that *DeepConvLSTM* is superior compared to any of the three benchmark models in terms of loss when training on either day sets. By looking at just the losses, ConvLSTM was second lowest in terms of MAE, whilst ARIMA is second lowest in terms of MAE. However, if we examine the predicted values, it reveals some hints as to why that is the case.

In Fig. 7, the predicted value by our model is simply a straight line over time. From the perspective of simply minimising loss, it is able to maintain minimal distance from the true absolute within-day returns. However, it is unable to capture any of the large fluctuations. Such a forecast is not reliable since it does not capture any of the large volatile movement at all. Arguably, having to capture those movements may be fairly important. On the other hand, if we consider the predictions based on the ARIMA model, as shown in Fig. 8, we can see that it is able to follow the fluctuations a bit more. Unforetunately, in most cases, it is simply considering the lag from last period's value as a strong indicator of the coming period's absolute within-day returns. This means that the prediction of ARIMA has lag itself. Nonetheless, we see that when compared against deep learning-based benchmarks or tradtional time series forecasting methods, *DeepConvLSTM* still has good forecasting power, albeit not particularly flexible in forecasting for large ups and downs.

4.2 Cross-Validation

Instead of only relying on the loss based on a single partition of the data into training, validation, testing, we want to consider different combinations. Namely, we we will do is to consider 2 other sets of data to train on and effectively do a k-fold cross-validation based on the temporal sequencing. That is, we first consider only the data from the early parts of the full data. The second set of data would then re-introduce slightly more data, whilst shift the validation and testing backwards in the time series. The third set (full set) will again introduce more data, and shift the validation and

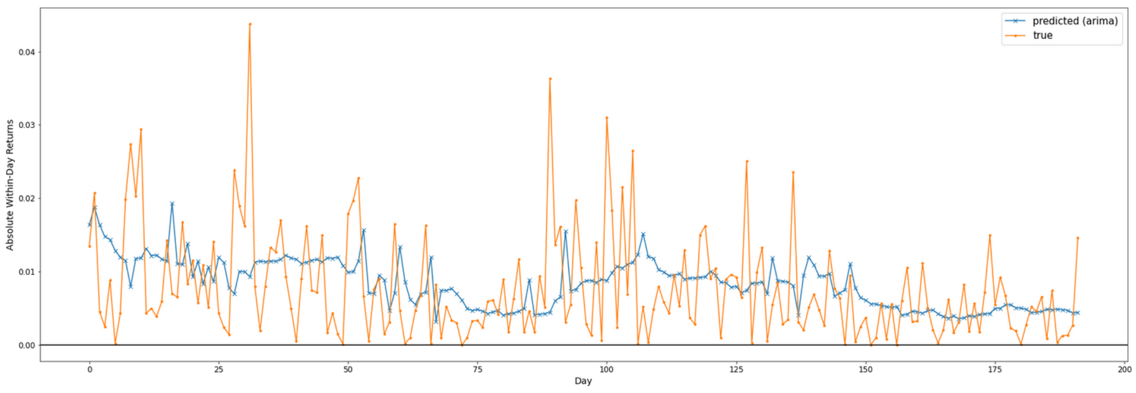

Figure 8: Predicted Values by ARIMA on US500 Data

Data Part	**Training End Date**	**Validation End Date**	**Testing End Date**
	US500	US500	US500
1st (T=896)	30/Oct/15	31/Oct/16	03/Aug/17
2nd (T=1344)	03/Aug/17	31/Oct/16	06/May/19
3rd (T = 1792 Full Data)	06/May/19	01/May/20	28/Jan/21

Table 2: Dates of Data Split

testing data back. Table 2 shows which dates corresponded to the end of which data set. Note that due to time and computational constraint, this was only possible to be completed on the US500 data set and not the UK100 market index.

Now by training each of the models on each of the three sets of data, we yield the results reported in Table 3. Where the value underlined indicates that it is the minimum loss among the rest within its category, we see that *DeepConvLSTM* excelled compared to the other models. In most of the categories, including the average of the three, the model still holds the strongest forecasting power. By undergoing such procedure to cross validate the model, it provides extra insurance that the model can perform.

4.3 Hyperparmeter Tuning

Although we do not have the liberty in time and a surplus of computing capacity, we were still able to consider the tuning of the number of ConvLSTM-CNN blocks that we incorporate within *DeepConvLSTM*. By training 2 other models, one which only contains only 2 blocks and the other 4 blocks, we can explore how varying with the number of blocks and the depth of the network may impact the results. Based on Fig. 9, we can see that there is not clear relationship for MAE over the number of blocks within the network. However, the graph corresponding to RMSE suggests that there may be a weak positive relationship in the RMSE and the number of blocks used. If this is valid, it may well threaten the validity of our model which advocates for neural networks that are

Data Part (US500)	***DeepConvLSTM***		**ConvLSTM**		**ARIMA**		**1D CNN-LSTM**		***DeepConvLSTM* + Max & Min**	
	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE
1st	$\underline{0.0380}$	$\underline{0.0277}$	0.0609	0.0500	0.1221	0.0803	0.1294	0.1179	0.0515	0.0461
2nd	$\underline{0.0872}$	0.0594	0.0962	0.0709	0.1198	0.0847	0.2597	0.2387	0.0907	$\underline{0.0567}$
3rd (All Data)	$\underline{0.1102}$	$\underline{0.0724}$	0.1196	0.0802	0.1129	0.0862	0.1169	0.0993	0.1186	0.0775
Average	$\underline{0.0785}$	$\underline{0.0532}$	0.0922	0.0670	0.1183	0.0838	0.1673	0.1520	0.0870	0.0601

Table 3: US500 Cross-Validation Loss

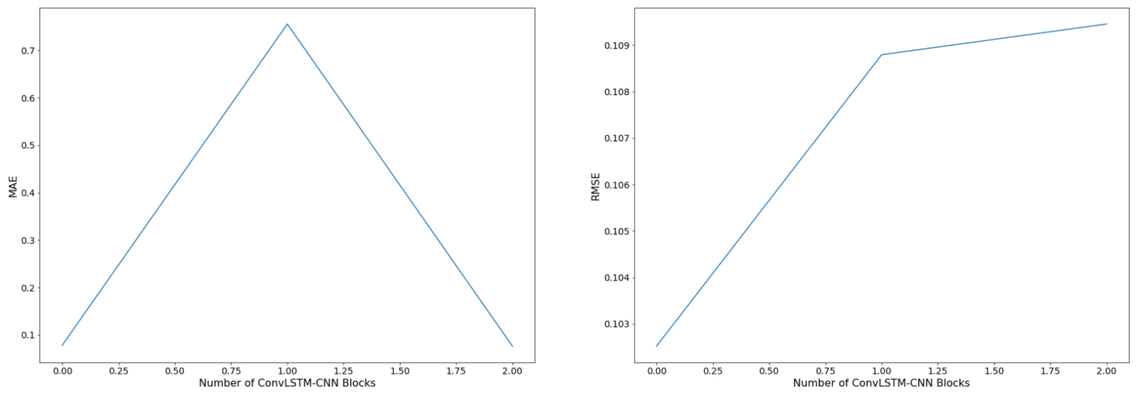

Figure 9: Losses for *DeepConvLSTM* with Varying Number of ConvLSTM-CNN Blocks

deeper. Nevertheless, this is not strong evidence, due to the inconclusive evidence with the MAE plot, but also too few data points were collected. Again this would only be possible should there be more computational resource to spare.

4.4 Limitations

Although the comparison of our *DeepConvLSTM* model against other models has concluded that it is generally the better model at forecasting volatility in the form of absolute within-day returns, it is still far from flawless. As mentioned already, the results that the model forecasts is not able to fully capture large fluctuations. Just as the issue with the validation data being heavily skewed due to the occurrence of a global health crisis, our model is not capable of dealing with unfamiliar situations and patterns. Of course, this criticism may be applied to the field of Deep Learning in general. However, given the context of the problem we aim to solve is to forecast and predict the volatility of the stock market next day, having to face a great deal of uncertainty is inevitable. Hence, it is important for us to try to overcome this limitation, possibly by devising more robust methods, such as using maybe an alternative loss functions.

Furthermore, should be noted that the run times of the models is fairly important too. With image conversion techniques, it immediately scales up the number of data points to process in a quadratic manner. Moreover, when training for the US500 data, compared to the ConvLSTM model, which only had around 42,000 trainable parameters, *DeepConvLSTM* had significantly more parameters to train, i.e. around 250,000. Together this may lead to problems in terms of whether the model architecture can scale to large samples or not. This may be especially necessary, since financial data can be very high-frequency. If we choose to use this model to train on all of it, the computational burden can be massive. Another threat to our findings is that we were unable to measure the runtimes of model effectively. Since the majority of the programming was completed on Google Colab using its complementary GPU resource, as it was used more, the quality of the GPU deteriorated. Thus, we cannot obtain fair measures of training time as it varies often between sessions.

4.5 Model Extension

Before we conclude, we want to provide an idea for extending this neural network structure. As we build on the idea of image conversion, we want to utilise the fact that any time series that is projected onto the 0 to 1 interval can only be completely characterised by its image form when information about the time series' maximum and minimum values are known. Essentially, these two values are the keys to inverting the GAF image. Therefore, when design the model around doing sequential forecasting on time series as images, we may want to embed the information of the maximum and the minimum values into the input of the network as well. As a result, we have devised the following extension of the *DeepConvLSTM* – called *DeepConvLSTM* + Max-Min.

Aside from the core structure of the *DeepConvLSTM* being the stacked ConvLSTM-CNN blocks, the *DeepConvLSTM* + Max-Min network would additional take 3 other inputs: The GAF image representation of the time series of the minimum prices over the period of the original input subsequence, the GAF image representation of the time series of the maximum prices over the period of

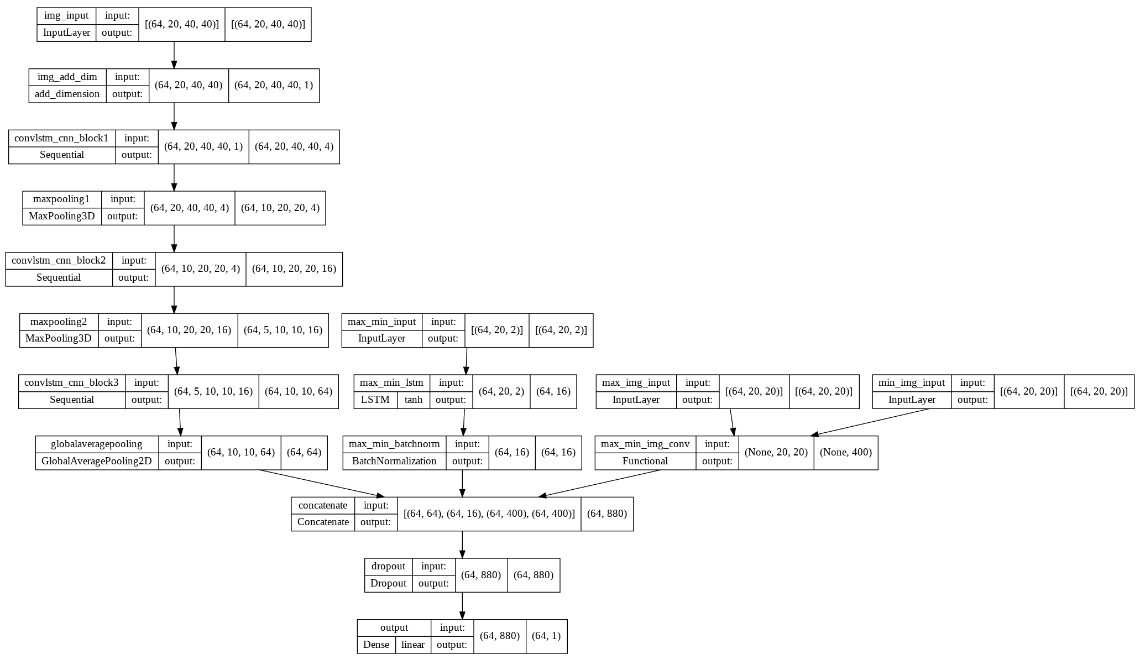
img_input InputLayer input: output: [(64, 20, 40, 40)] [(64, 20, 40, 40)]
img_add_dim add_dimension input: output: (64, 20, 40, 40) (64, 20, 40, 40, 1)
convlstm_cnn_block1 Sequential input: output: (64, 20, 40, 40, 1) (64, 20, 40, 40, 4)
maxpooling1 MaxPooling3D input: output: (64, 20, 40, 40, 4) (64, 10, 20, 20, 4)
convlstm_cnn_block2 Sequential input: output: (64, 10, 20, 20, 4) (64, 10, 20, 20, 16)
maxpooling2 MaxPooling3D input: output: (64, 10, 20, 20, 16) (64, 5, 10, 10, 16)
max_min_input InputLayer input: output: [(64, 20, 2)] [(64, 20, 2)]
convlstm_cnn_block3 Sequential input: output: (64, 5, 10, 10, 16) (64, 10, 10, 64)
max_min_lstm LSTM tanh input: output: (64, 20, 2) (64, 16)
max_img_input InputLayer input: output: [(64, 20, 20)] [(64, 20, 20)]
min_img_input InputLayer input: output: [(64, 20, 20)] [(64, 20, 20)]
globalaveragepooling GlobalAveragePooling2D input: output: (64, 10, 10, 64) (64, 64)
max_min_batchnorm BatchNormalization input: output: (64, 16) (64, 16)
max_min_img_conv Functional input: output: (None, 20, 20) (None, 400)
concatenate Concatenate input: output: [(64, 64), (64, 16), (64, 400), (64, 400)] (64, 880)
dropout Dropout input: output: (64, 880) (64, 880)
output Dense linear input: output: (64, 880) (64, 1)

Figure 10: *DeepConvLSTM* + Max-Min Overall Architecture

the original input subsequence, and lastly, the actual subsequence of the maximum and minimum values themselves. The two GAF images will input into a series of 2-dimensional CNN to extract temporal features from them. The maximum and minimum value is to provide information regarding the absolute market index value which the orgiinal input GAF images represents. This vector of maximum of minimum values can directly go through an LSTM structure to forecast for its future values, which may provide extra information for forecasting the absolute within-day returns for the next day. This architecture is displayed by Fig. 10

By going through the same training, testing, cross-validation procedure as we did for all the other models, we yield decent results for this more extensive model. We start off by viewing the training & validation loss for the model, as shown in Fig. 11. We can see that the model behaves similar to the way *DeepConvLSTM* does, where the training loss converges smoothly, but the validation loss does not fall at a similar rate. But again, this is likely the consequence of the validation data being to tough to make forecasts on, due to the unforeseen impact of COVID-19 on the stock market. Next, if we consider Fig. 12, see that unlike *DeepConvLSTM*, this augmented version of the model is able to show more fluctuations in its forecasts around a certain range. This does not translate to strong predictive power necessarily, since based on Table 1, this model is not strictly better than *DeepConvLSTM* in achieving low losses. Yet, it shows some potential for the model to be more flexible in forecasting sharper peaks and larger degree of volatility.

Lastly, if we inspect Table 3, we can see that *DeepConvLSTM* + Max-Min is still better than the other benchmark models in most areas. Although it is inferior to *DeepConvLSTM* in most cases, the results for this augmented version of the model is still inspiring and is exciting to consider further extensions of the model in the direction of having a neural network that branches off multiple times. (An interesting idea might be to have a time series neural netowrk where each branch may represent a specific frequency of the data and that high-frequency data branches can be engineered to be inputs of the low-frequency data branch)

5 Conclusion

In conclusion, in this paper we designed the *DeepConvLSTM* in an attempt to forecast the stock index's volatility in the upcoming day. By adopoting Gramian Angular Field conversion and multiple ConvLSTM-CNN blocks, the model was delivered great results on the US500 and UK100 index. It achieved lower RMSE and MAE losses compared to benchmark models such as ARIMA and single layer ConvLSTM. That being said, the model still showed substantial limitations in its forecasting

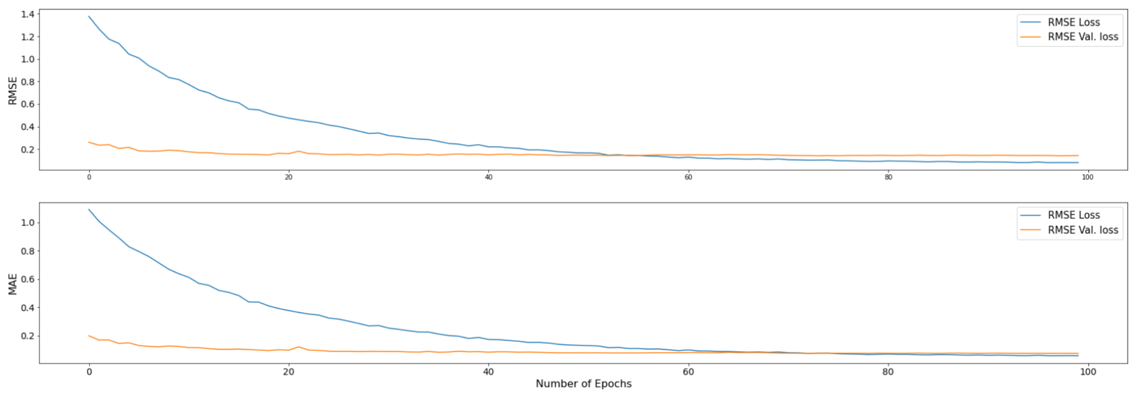

Figure 11: Loss for *DeepConvLSTM* + Max-Min Model

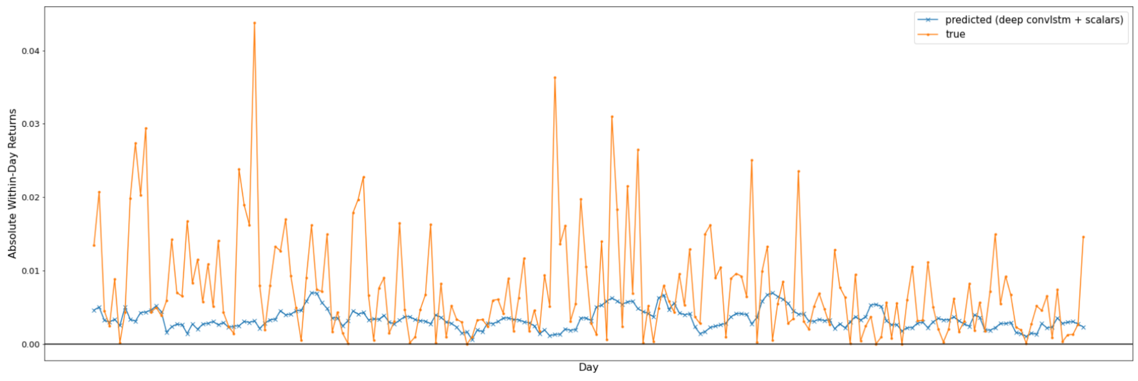

Figure 12: Predicted Values by *DeepConvLSTM* on US500 Data

power. Its forecasts were still far from the truth and is fairly unreliable for investors to plan their trading based off it, especially cosnidering the stakes in stocks trading. Part of the reason, is the fundamental difficulty of when forecasting financial data. However, we believe that it is very possible to further tune the model and to improve its predicative capability. This was unfortunately not possible for us due to constraints on time and computation resources.

Nonetheless, our findings naturally lead to various paths for potential further research. To begin with, let us mention some ways to improve our experiment and model itself, should there be less constraints on time and computation resource. Firslty, model tuning with the hyperparameters should be introduced to obtain a more robust model. For instance, investigating whether varying the optimiser or batch size may alter the model results. Secondly, the stock index time series data we looked at in this paper recorded prices at a 10 minute interval. This is nothing close to the highest frequency of data that stock prices are recorded at. This aspect weakens the applicability of our results in areas like high-frequency trading. If we looked at higher frequency data, the number of data points contained within each day would increase correspondingly. Subsequently, the GAF images that we may generate for each day would contain more of the temporal dependencies and correlations within each day. Yet, this is only possible should there be more powerful computing power to train the model, as the image conversion increases the size of the data in a quadratic manner.

Next, throughout our experiments, we have used standard deviation as a proxy for the volatility of the stock index. It is the most basic, and arguably sub-optimal, way to capture the notion of volatility using a quantitative measure. There are of course many other possible representations of volatility. For instance, absolute returns of stocks was proposed to be a better alternative to measure volatility (Ding et al., 1993). Therefore, how to tune and adapt similar models for forecasting different measures of volatility may require attention. In another sense, trying to get a composite forecast for volatility by devising models on several measures collectively can be fruitful.

A more ambitious problem to tackle using similar techniques might be to predict the entire time series within the next day, which then allows for a more primitive approach to forecast the daily volatility, as we would already know what all the data points will be during the next day. Furthermore, although we only treated absolute within-day returns as an medium that captures volatility, forecasting within-day returns with the correct might offer more information about volatility, and on its own not an easy problem too. in many cases, the large volatility in both directions are not well captured by the predictions. Predictions would likely centred around 0 to cope with the polar fluctuations as they aim to balance the loss in both sides. Therefore, this asks for possibly further work done in devising a more robust loss function, or to seek for other routes to modelling the problem. Possibly models that combines the forecast of the magnitude and the binary forecast of the sign next day would be effective.

Another direction one may consider would be to construct forecasting models using similar genre of techniques. Our work provides confidence in the effectiveness of techniques that convert time series into images and using them for Deep Learning modelling. Indeed GAF is a good option, but there are alternative options that bring the same function. For instance, Markov Transition Fields is another valid technique. It has been shown effective at classification tasks, e.g. online fraud detection (Zhang et al., 2018), vibration events in fibre optics (Zhao et al., 2022), etc. However, little work has been done on using it for time series forecasting purposes. Therefore, this may be a possible area to explore.

Furthermore, aligning with the trend of transformer-based Deep Learning models, it is also worth investigating application of such models for financial forecasting problems. Naturally, transformers that are designed for time series are good candidates to examine. Wen et. al. (2022) has compiled a survey for recent time series-based transformers, and would be a good starting point to seek for inspiration. A particular genre of transformers that might be more relevant to our work in this paper would be transformers for video prediction problems. ConvLSTM has been a common network structure used in video prediction. An example of this would be the attempt by Mukherjee et. al. (2019) in predicting the next frame of a video based on previous frames. Consequently, it is exciting to see whether techniques applied to video prediction can combine well with image conversion to do time series forecasting effectively. The ViViT transformer-based model, recently proposed by Arnab et. al. (2021) may fulfill this purpose.

References

Ding, Z., Granger, C. W., & Engle, R. F. (1993). A long memory property of stock market returns and a new model. *Journal of empirical finance*, *1*(1), 83–106.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, *9*(8), 1735–1780.

Gers, F. A., & Schmidhuber, J. (2000). Recurrent nets that time and count. *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium*, *3*, 189–194.

Malkiel, B. G. (2003). The efficient market hypothesis and its critics. *Journal of economic perspectives*, *17*(1), 59–82.

Poon, S.-H., & Granger, C. W. (2003). Forecasting volatility in financial markets: A review. *Journal of economic literature*, *41*(2), 478–539.

Aliev, R. A., Fazlollahi, B., & Aliev, R. R. (2004). *Soft computing and its applications in business and economics* (Vol. 157). Springer Science & Business Media.

Lin, M., Chen, Q., & Yan, S. (2013). Network in network. *arXiv preprint arXiv:1312.4400.*

Chollet, F. et al. (2015). Keras.

He, K., Zhang, X., Ren, S., & Sun, J. (2015). Deep residual learning for image recognition. https://doi.org/10.48550/ARXIV.1512.03385

Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *International conference on machine learning*, 448–456.

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Jia, Y., Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, … Xiaoqiang Zheng. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems [Software available from tensorflow.org]. https://www.tensorflow.org/

Schmidhuber, J. (2015). Deep learning in neural networks: An overview. *Neural networks*, *61*, 85–117.

Shi, X., Chen, Z., Wang, H., Yeung, D.-Y., Wong, W.-K., & Woo, W.-c. (2015). Convolutional lstm network: A machine learning approach for precipitation nowcasting. *Advances in neural information processing systems*, *28*.

Wang, Z., & Oates, T. (2015). Imaging time-series to improve classification and imputation. https://doi.org/10.48550/ARXIV.1506.00327

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT press.

Keskar, N. S., Mudigere, D., Nocedal, J., Smelyanskiy, M., & Tang, P. T. P. (2016). On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836.*

Zhang, R., Zheng, F., & Min, W. (2018). Sequential behavioral data processing using deep learning and the markov transition field in online fraud detection. *arXiv preprint arXiv:1808.05329.*

Bhowmik, R., & Wang, S. (2020). Stock market volatility and return analysis: A systematic literature review. *Entropy*, *22*(5), 522.

Dhillon, A., & Verma, G. K. (2020). Convolutional neural network: A review of models, methodologies and applications to object detection. *Progress in Artificial Intelligence*, *9*(2), 85–112.

Faouzi, J., & Janati, H. (2020). Pyts: A python package for time series classification. *Journal of Machine Learning Research*, *21*(46), 1–6. http://jmlr.org/papers/v21/19-763.html

Lu, W., Li, J., Li, Y., Sun, A., & Wang, J. (2020). A cnn-lstm-based model to forecast stock prices. *Complexity*, *2020*.

Moghar, A., & Hamiche, M. (2020). Stock market prediction using lstm recurrent neural network. *Procedia Computer Science*, *170*, 1168–1173.

Rasheed, J., Jamil, A., Ali Hameed, A., Ilyas, M., Özyavaş, A., & Ajlouni, N. (2020). Improving stock prediction accuracy using cnn and lstm. *2020 International Conference on Data Analytics for Business and Industry: Way Towards a Sustainable Economy (ICDABI)*, 1–5. https://doi.org/10.1109/ICDABI51230.2020.9325597

Hassan, R. J., Abdulazeez, A. M. et al. (2021). Deep learning convolutional neural network for face recognition: A review. *International Journal of Science and Business*, *5*(2), 114–127.

Zhao, X., Sun, H., Lin, B., Zhao, H., Niu, Y., Zhong, X., Wang, Y., Zhao, Y., Meng, F., Ding, J., Zhang, X., Dong, L., & Liang, S. (2022). Markov transition fields and deep learning-based event-classification and vibration-frequency measurement for ϕ-otdr. *IEEE Sensors Journal*, *22*(4), 3348–3357. https://doi.org/10.1109/JSEN.2021.3137006

A Hyperparameter Settings

Hyperparameter/ Setting	***DeepConvLSTM***	**ConvLSTM**	**1D CNN-LSTM**	***DeepConvLSTM* + Max & Min**
Batch Size	64	64	64	64
Learning Rate	1e-4	5e-4	5e-4	3e-4
Number of Epochs	100	100	10	100
Number of Total [Trainable] Parameters (US500)	253,677 [253,341]	42,001 [41,937]	42,225 [42,161]	312,701 [312,293]
Number of Total [Trainable] Parameters (UK100)	253,677 [253,341]	59,665 [59,601]	54,513 [54,449]	257,133 [256,725]
Optimiser	Adam	Adam	Adam	Adam
Loss Function	MSE	MSE	MSE	MSE
Early Stopping Patience (During Cross-Validation)	20	15	5	20

Table 4: Hyperparameter and Training Settings

B Contributions

dydx201822139: Preprocessed the data, designed most of the models, generated most of the results, wrote up the majority of the report

azure-fog-201706586: Designed and created one of the models, generated results and plots for some of the results

BusyApe201804930: Collected the data, cleaned some of the data, completed literature review