Forecasting Financial Time Series with CNNs

AIMS CDT Mini Project

Supervised by Prof. Stephen Roberts

Fabian Fuchs

12/06/2017

Abstract

A convolutional neural network architecture is presented for financial time series forecasting. Historical data of daily open, close, high and low prices as well as traded volume from 20 stocks is used to forecast the trend of a single stock for the next day. The out-of-sample performance averages at 50.9%, which is highly significant for financial data. Very high accuracies for specific stocks as well as a large number of suggested optimisation approaches for future work show the potential of this algorithm. A range of visualisation techniques is provided to enable further analysis on the performance and optimisation of the algorithm.

1 Introduction

Predicting stock prices is both a notoriously challenging and intriguing task. For investors, predictive prowess increases expected returns. From a machine learning research perspective, forecasting stock prices is a particularly interesting challenge due to the vast amount of data in which patterns are equally abundant and hard to detect. Due to its noisy, random-walk like nature, it is one of the most difficult and therefore also one of the most intriguing problems in the field of signal processing.

For investors, analysis of stocks can be divided into two parts [1]. First, the current performance and the future potential of the company, the industry and the economy is assessed. Second, the development of the stock price itself is analysed. The second part, which is called technical analysis, relies purely on the recent history of the stock prices themselves, their trends and statistics. It does not attempt to estimate the intrinsic value of a stock, but purely relies on the detection of patterns in the behaviour of the market [1]. In our work, we focus on the second part. We use daily stock prices (open, close, high, low) and trading volume of the past 100 days from different stocks simultaneously to predict the next day closing price of one specific stock.

Applying neural networks to financial time series forecasting is not a new idea. Already in the 1990's, during the second wave of artificial neural networks research [2], ANNs were employed in this domain [3, 4, 5]. In fact, the perceived potential was so high that financial services institutions were the second largest sponsors for research in ANN applications in the early 1990's [6, 7].

While in the 1990's wave, researchers mostly applied simple and small dense neural networks, today, however, more complex and sophisticated architectures are chosen. Specifically, recursive models such as LSTMs attract a lot more attention [9]. Convolutional neural networks, which play a big role in the success of computer vision [2], are less popular in the field of financial time series forecasting. In computer vision, one of their main contribution to the network's understanding of the content of the image is detecting edges. We believe that this extraction of local differences has potential in detecting patterns in financial time series as well when the input of the neural network is represented as an image. To that end, we take the stock data of 20 different stocks from the last 100 days, stack them on

top of each other, and use this two dimensional array as input for the CNN.

Neural networks are being used to perform both regression [10] and categorization [8] in the domain of forecasting financial time series. We decided to perform categorical forecasts for three reasons: First, neural networks in general perform better on classification tasks (see computer vision) than on regression tasks. Second, the performance is much more easily measured and comparable, e.g. when looking at different equities. Third, an investor is primarily interested in whether a certain index goes up or down and not so much in its precise value. In our case, we predict classes of 'up' and 'down', referring to whether the next day close price is above or below the current closing price. However, this system can easily be augmented, for example with a third class termed 'roughly constant'.

In section 2, the data set and the preprocessing steps are introduced. In section 3, the network architecture is explained. Section 4 presents the results obtained in this work and in section 5, conclusions are drawn and future work is discussed.

2 The Data Set

The overall data set comprises 5000 days of trading data from the S&P500 index. The data set has been split into training data (1994/01/03 to 2007/04/23) and test data (2007/04/24 to 2013/11/07). Hence, two thirds of the data set were used for training the network, one third was held out for testing the performance of the algorithm.

Out of the over 500 (partly temporary) constituents of the index, 20 were chosen at random. The only selection criterion was that every stock included in the data set had to have been a permanent constituent of the S&P500 from 1994 to 2013. In future work, we will examine different strategic selections of stocks. Possible strategies include only looking into one industry at once or including the largest representative from each industry. However, in this work, we wanted to include a random selection to then examine the information flow and see whether this matches the intuition one might have about interdependence of certain stocks.

For each stock and trading day, five data points were included in the data set: opening price, closing price, daily high, daily low, and traded volume (all measured in USD). The first four values are used for plotting and analysing 'candlestick

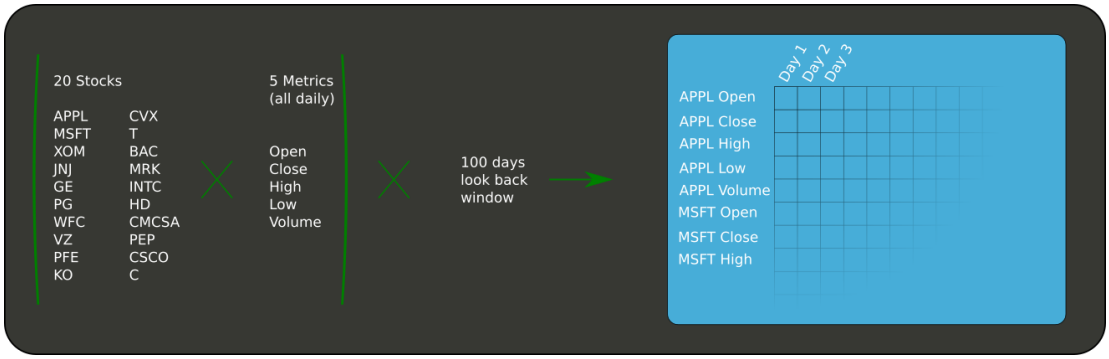


Figure 1: Visualisation of the input the neural network receives for predicting the next day closing price of one specific stock. The network receives a 100 times 100 matrix where one dimension is time (100 days) and the other is made up from 20 stocks with 5 different metrics each.

data', a popular tool for technical analysts with controversial predictive prowess [8]. These four data streams are augmented by the trading volume to enable the algorithm to detect additional patterns related to trading activities over time. This type of data set is very typical, easy to obtain and also used by many other researchers in this field (e.g. [8]). However, in our case, the algorithm does not only have the data from the last few days as input (compare with e.g. [8]), but the data from the last 100 days. The algorithm then uses this data to predict whether the closing price of a specific stock on the next day is higher or lower than on the current day. The input of the neural network is visualised in figure 1.

Many researchers in this area include other technical data as input for their learning algorithm such as the 10 day moving average [1, 10] or the relative strength index [1, 10, 11]. However, since the input in our case is the data from the last 100 days, the network could potentially learn its own technical indicators. Nevertheless, this algorithm can easily be augmented by additional input and it can even be used to visualise which inputs were most important for the predictions the algorithm made.

3 Network Architecture

The chosen architecture (visualised in figure 6) consists of one convolutional layer followed by two fully connected layers. The first fully connected layer has 32

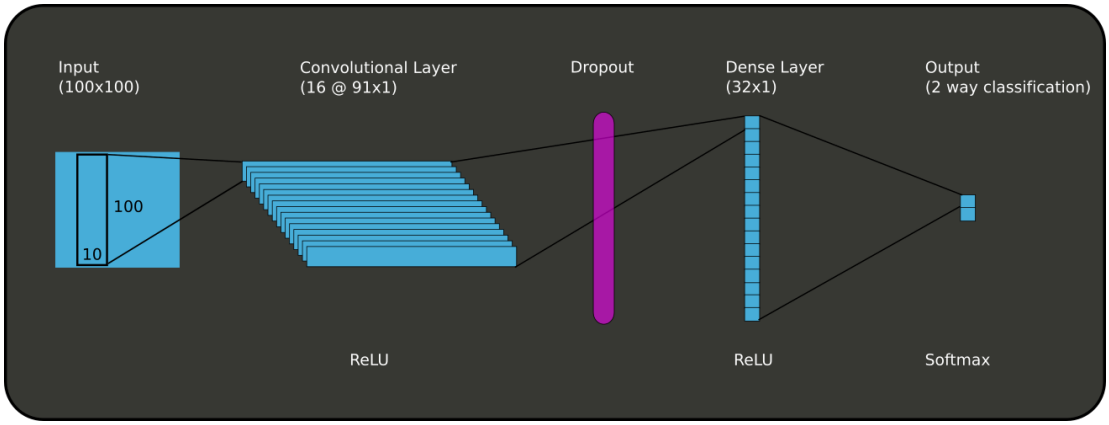


Figure 2: Visualisation of the neural network architecture. The bottom line details the activation function used in the respective layer.

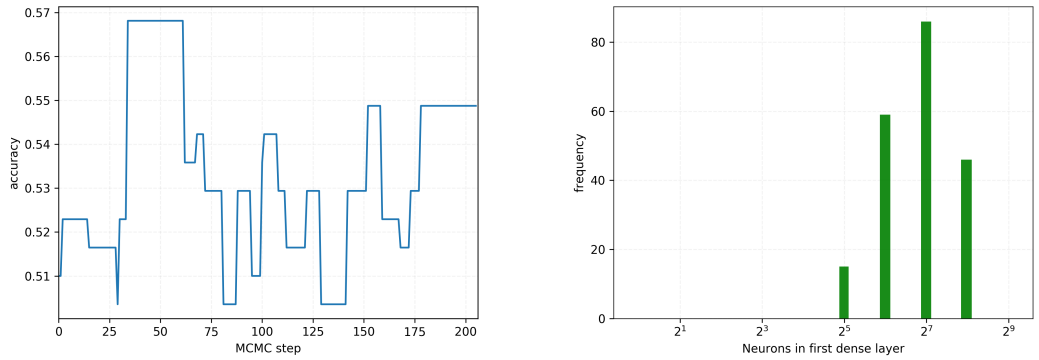
neurons, the second is the output layer which has two neurons for two way classification. The dimensions of the convolutional filter is 10 (in the time domain) times 100 (in the stock index domain). Hence, the filter looks at all stocks and all metrics at once but only at 10 days of data at a time. It then performs its 1D convolution in the time domain with stride 1. The convolutional layer has 16 channels which proofed to work significantly better than lower numbers of channels.

3.1 Extreme Early Stopping and Seed-Averaging

Due to the relatively complex network with a high number of parameters compared to the small data set, the algorithm is prone to overfitting. Even though a dropout of 40% was applied between the two hidden layers, the algorithm starts overfitting even after very small number of epochs. Small degrees of overfitting are not necessarily harmful for the test performance and are often necessary especially in finance, where the true patterns are notoriously hard to detect. One could say that there is only a chance that the algorithm detects true patterns if it is allowed to pick up false patterns as well. In our case, we decided to stop the training after the training accuracy reached about 62% (compared to roughly 50% test accuracy). This was the case after 20 epochs. Since the number of epochs is very small, the network is nowhere near convergence. Hence, we applied a novel approach of training the algorithm hundreds of times with different weight initial-

isations and then combining the predictions. In order to get very stable results with high statistical significance, we trained the algorithm 400 times on the same data.

3.2 MCMC Optimisation



(a) **(b)**

Figure 3: Optimising the hyperparameters of the neural network with MCMC. (a) shows the performance per MCMC step. (b) shows the distribution obtained against the number of neurons in the first dense layer.

One of the most challenging tasks when applying a neural network to a given problem is to optimize its hyperparameters. This can be done either manually or by applying an optimization algorithm such as a Markov Chain Monte Carlo Metropolis Hastings algorithm. The MCMC algorithm was developed to explore functions, more specifically their optima, which have the characteristics of a probability density function. If being run for infinite time, it is guaranteed to explore all areas of the function and the amount of time it spends at a certain location in parameter space is proportional to the value of the function [13]. We defined this function to be the accuracy of the top 1% predictions minus 0.5 (for an explanation of what is meant by 'top 1% predictions', see section 4.1). Whenever a set of hyperparameters led to a negative value (performance below 50%), the step was rejected instantly. In general, this is a very interesting approach to optimising neural networks, especially due to the high dimensionality of the hyperparameter space. However, as our network became more and more complex over the course

of the project, the time needed for evaluating one architecture exceeded 24 hours, which is much too long for MCMC to work. Hence, the same optimisation was done manually. In order to still be able to deliver a proof of concept, we combined the results not of 400 different runs, but of 20 for each of the MCMC steps. Figure 3 shows the results of running the MCMC for approximately 2 weeks. Figure 3a shows a typical MCMC trajectory: The algorithm prefers parameter sets which lead to high values of the target function (accuracy), but it also sometimes accepts parameters which lead to a decreasing performance. Figure 3b shows the results for the number of neurons in the first dense layer. The MCMC suggests that 128 neurons is the ideal size. However, manual optimization showed that 32 neurons lead to the best results. The reason for this discrepancy is most likely the low number of runs used in MCMC and the resulting lack of statistical significance of the obtained accuracy.

3.3 Manual Optimisation

As stated above, in order to obtain statistically significant results, the algorithm had to be run for more than 24 hours. Therefore, optimisation had to be done manually. Table 1 shows the results for a selection of hyperparameters. In this iteration of optimisation, the network was started with the following set of parameters: batch size 10, epochs 20, optimiser Adadelta, learning rate 0.001, dropout 1st layer 0%, dropout 2nd layer 40%, channels in convolutional layer 4, stride 1, neurons in first dense layer 32. According to the results seen in the table, only the number of channels was changed, from 4 to 16. This set of hyperparameter values is used for all results shown in this report. However, since changing one hyperparameter can have significant impact on the optimal value of another parameter, more iterations of optimisation will be necessary in future work to further fine tune the performance of the network. In this specific case for example, it is likely that increasing the number of channels increases the optimal dropout ratio as well as the optimal number of neurons in the dense layer.

Table 1: Manual optimisation of hyperparameters. The table shows the results for varying values of hyperparameters.

Dropout 1st Layer	0%	10%	20%
Performance	53.8%	53.3%	53.5%
Dropout 2nd Layer	20%	40%	60%
Performance	52.2%	53.8%	53.5%
Channels in Conv. Layer	4	8	16
Performance	53.8%	55.8%	57.8%
Stride (for Convolution)	1	5	10
Performance	53.8%	52.1%	51.2%
Neurons in First DL	16	32	128
Performance	53.2%	53.8%	52.8%

4 Results

4.1 Performance on different Stocks

In figure 4, the results for predicting the next day closing price of the Apple stock are visualised. Shown are the accuracies of the prediction whether the next day closing price is above ('up') or below ('down') the closing price of the current day. For Apple, 49.1% of the training data and 52.9% of the test data were up-days. The confusion matrix (figure 4a) and the ROC curve (figure 4b) show that the algorithm performs just as badly as random guessing. The confusion matrix shows that the true label does not seem to have any impact on the predicted label and the ROC curve is a diagonal line showing that altering the threshold of classification doesn't change the proportion of the true positive rates over the false positive rate, which is approximately 1.0. This is neither particularly surprising nor worrying as it is notoriously difficult to get accuracies significantly above 50% when forecasting stock prices.

However, the strength of this network unfolds when ordering the predictions according to the certainty of the classification algorithm (figure 4c). As the network minimises the loss function, the output of the algorithm, which we refer to as

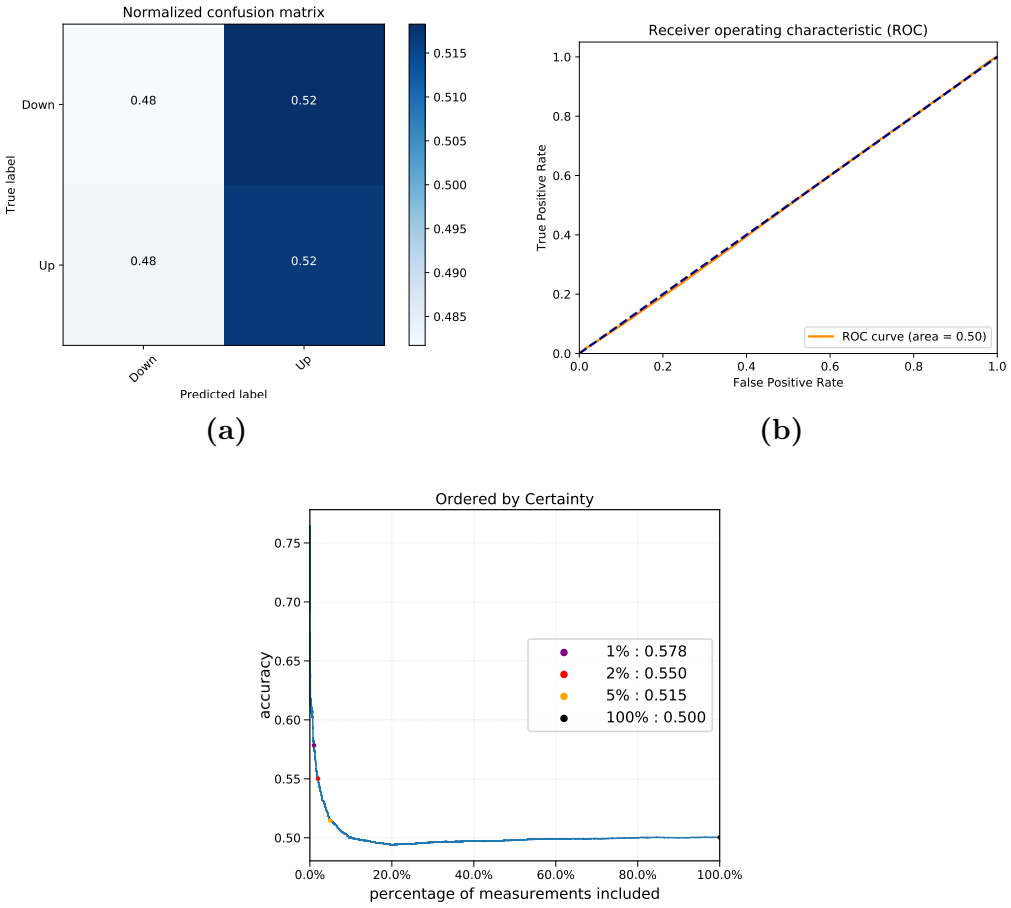


Figure 4: Results for predicting the next day closing price of the Apple stock. (a) shows the confusion matrix and (b) shows the ROC curve. In (c), the measurements are ordered according to how certain the algorithm was about its prediction.

'certainty', can be seen as the maximum a posteriori estimate. Whereas there is no advantage over random guessing if all predictions are included, there is a very significant advantage if only the top one or top two percent of the predictions are taken into account (57.8% and 55.0%, respectively). In fact, accuracies of this order of magnitude are considered exceptional when forecasting financial markets. It is interesting as well that there is a dip in performance below random guessing when including 20% of the predictions. This effect was observed for different architectures and different runs when forecasting the Apple stock, but not for

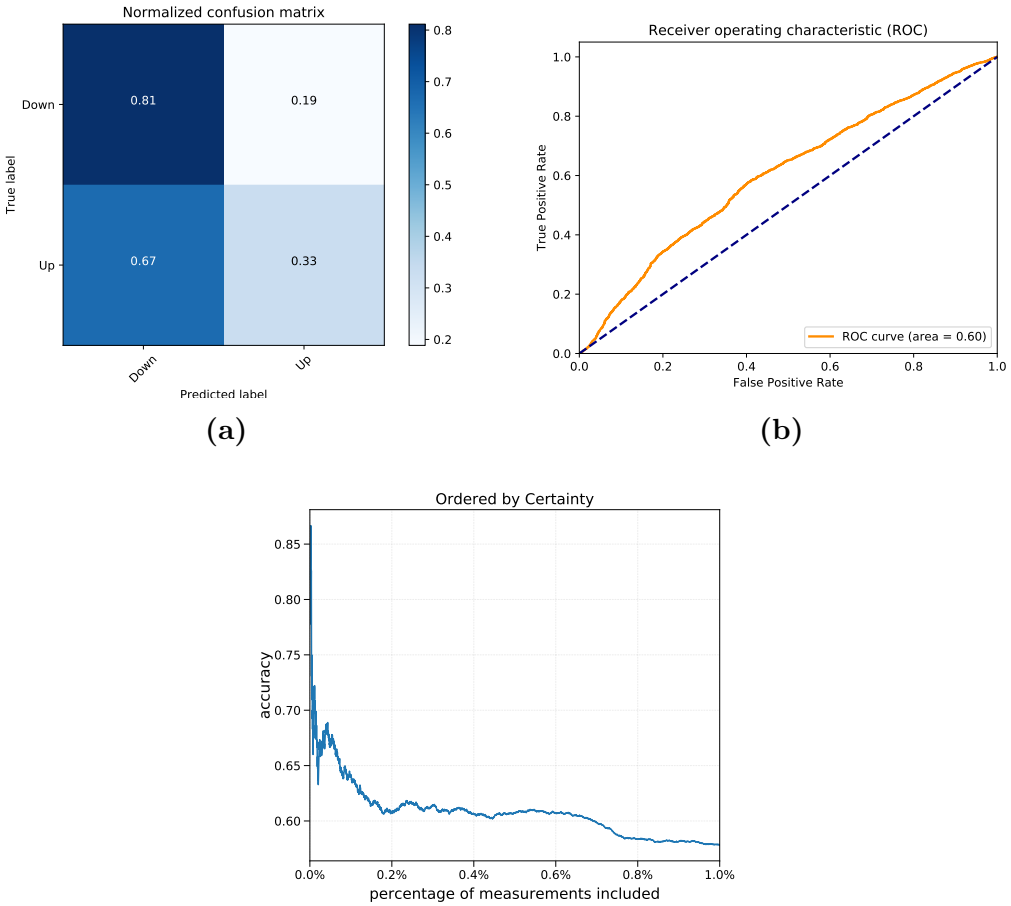


Figure 5: Top one percent of most certain predictions for the next day closing price of the Apple stock. (a) shows the confusion matrix and (b) shows the ROC curve. In (c), the measurements are ordered according to how certain the algorithm was about its prediction.

other stocks. Hence, this dip is most likely an artefact of the specific data set for Apple. To gain more insight into what the algorithm is doing and how it is performing for the very high certainty predictions, we zoom into the top one percent and analyse these predictions in figure 5. The ROC curve now has an area under curve of 0.60 and the confusion matrix shows that the predicted label has a strong correlation of predicted and true label. The bias towards 'down' predictions (figure 5a) might stem from the fact that there were more 'down' than 'up' days

in the training set. However, with 52.9% 'up' days in the test data set, this makes the obtained accuracy even more impressive.

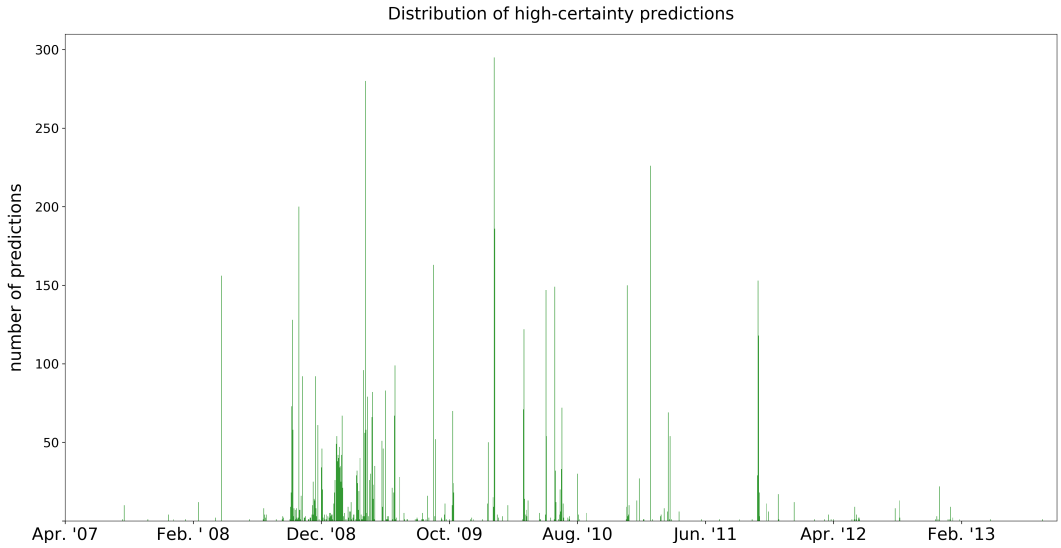


Figure 6: Locations in time space of top one percent predictions for the next day closing price of the Apple stock.

Figure 6 shows at which points in time the predictions with the highest certainty are made. As the results are combined from predictions of 400 runs, a single point in time could potentially have up to 400 counts meaning that all 400 independently trained networks predicted this specific day with an extremely high certainty. While there are days with 150 to 300 counts, a total of 284 days have been covered at least with one high-certainty prediction. Hence, with this technique, not just 1 percent, but 18% of all days are covered. Of course, this representation opens up a lot of possibilities for follow-up work and potential optimisation techniques.

Table 2 shows the accuracies for 20 different stocks. The algorithm was tested and optimised only on the Apple stock. Therefore, it is not surprising that the average performance on other stocks is lower than on Apple. Looking at the top 1% predictions, even the mean accuracy over all stocks of 50.9% is promising, given that there are still a lot of opportunities to optimise this algorithm (see section 5). However, the performance varies a lot between stocks (the standard deviation reads 7.1%), which means that the true potential of this algorithm lies in detecting why

Table 2: Accuracies for next day closing price prediction for different stocks. The measurements are ordered according to the certainty of the prediction and only the top x% are included (see 'Measurements Included').

	Measurements Included		
Stock	100%	2%	**1%**
AAPL	50.0%	55.0%	**57.8%**
BAC	48.7%	39.2%	**34.6%**
C	49.1%	44.0%	**42.8%**
CMCSA	49.8%	50.4%	**50.8%**
CSCO	50.3%	49.2%	**48.5%**
CVX	50.9%	52.7%	**55.7%**
GE	49.6%	44.3%	**45.3%**
HD	50.5%	47.4%	**46.3%**
INTC	50.2%	54.5%	**56.0%**
JNJ	49.7%	49.0%	**50.7%**
KO	49.2%	44.3%	**40.9%**
MRK	50.6%	54.7%	**55.2%**
MSFT	50.9%	58.7%	**62.5%**
PEP	49.6%	54.5%	**56.6%**
PFE	49.4%	48.6%	**48.7%**
PG	50.3%	54.7%	**57.4%**
T	50.8%	46.5%	**42.8%**
VZ	49.6%	51.8%	**52.5%**
WFC	49.7%	53.4%	**54.4%**
XOM	50.3%	55.7%	**58.0%**
mean	**50.0%**	**50.4%**	**50.9%**
std	**0.6%**	**5.0%**	**7.1%**

it works very badly on some stocks and either improve the performance on these or finding a way of detecting *a priori* whether a certain stock is easily predictable or not.

4.2 Filter weights

Neural networks are often being declared as black boxes, they achieve useful results but it is often difficult to comprehend how they came to their conclusions. While neural networks are certainly more complex than most other machine learning algorithms (our relatively small CNN already has in the order of 10^5 parameters), there are ways to visualize what the network learned. In this section, we examine how important each of the inputs is for the network. To this end, we look at the learned weights of the convolutional layer, which is applied directly to the input data. Figure 7 shows three different filters from a network which has been trained on the XOM stock. The input is arranged as shown in figure 1. XOM is the third stock, hence row 11 to 15 correspond to the daily Open, Close, High, Low, and Volume of XOM. The filter shows interesting structure proofing that the network detects patterns from different stocks. The filter in figure 7b seems to put a lot of emphasis on all the XOM related input data whereas the filter in figure 7a seems to especially interested in the closing prices of the XOM stock.

Next, in order to receive a broader picture, we square the weights of the learned filters and then average them in three dimensions: the 400 runs, the 16 channels, and the width of the filter in the time domain. We then average one more time, depending on what type of inputs we want to compare. If we examine which stocks are most relevant for the predictions (figure 8a), we average over the metrics. If we want to determine which metric is most important for the forecasting (figure 8a), we average over the stocks. In order to obtain a good estimate of the importance of the inputs, they have to be normalized carefully. Otherwise, inputs with high absolute values will lead to smaller weights, regardless of whether they are particularly important for the network or not. In our case, an additional layer of normalisation was needed. Unfortunately, this decreased the performance of the algorithm. A purer approach would be to plot the output of the convolutional layer while setting all inputs to zero, except, e.g., the closing prices. The activation of the neurons

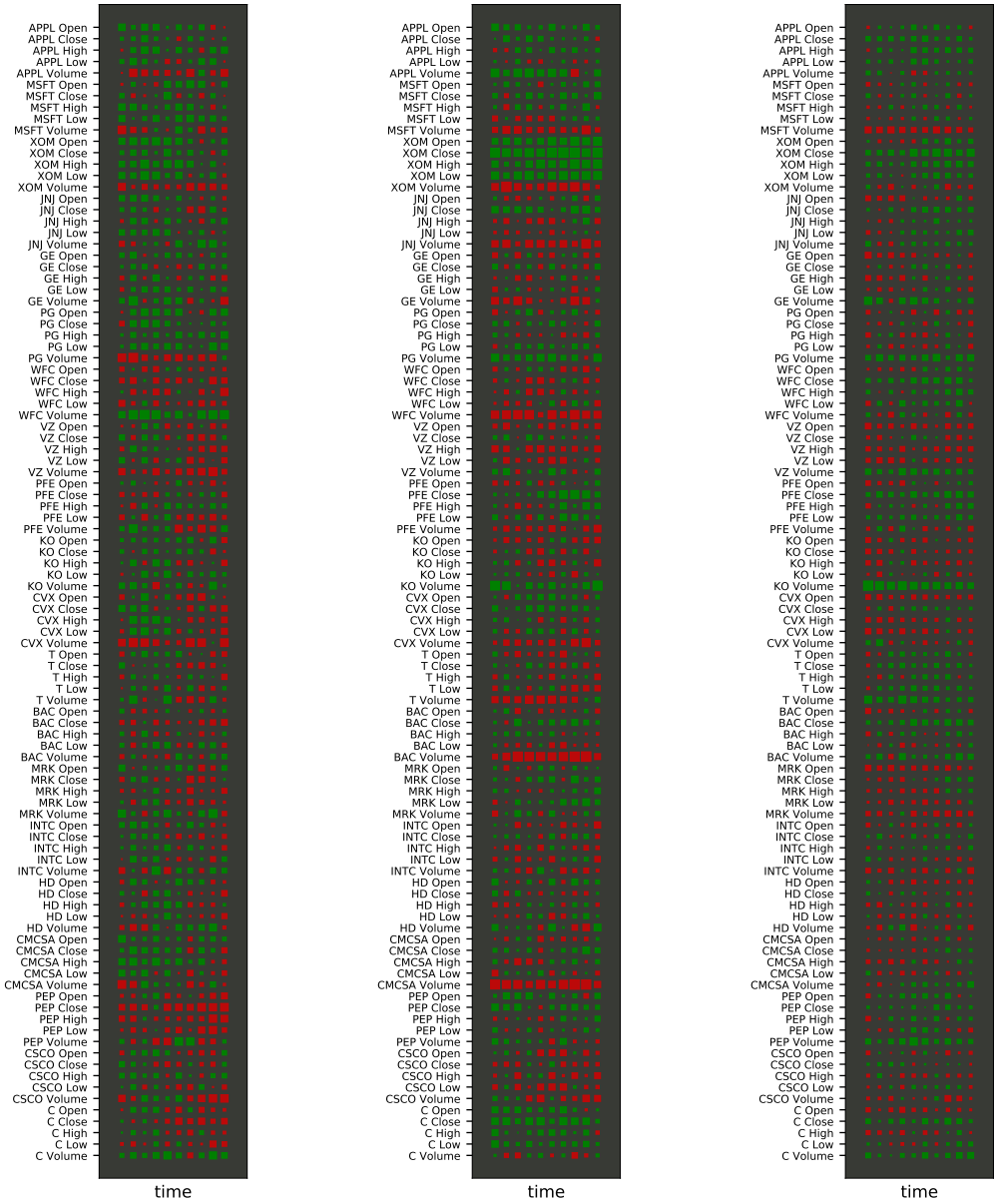
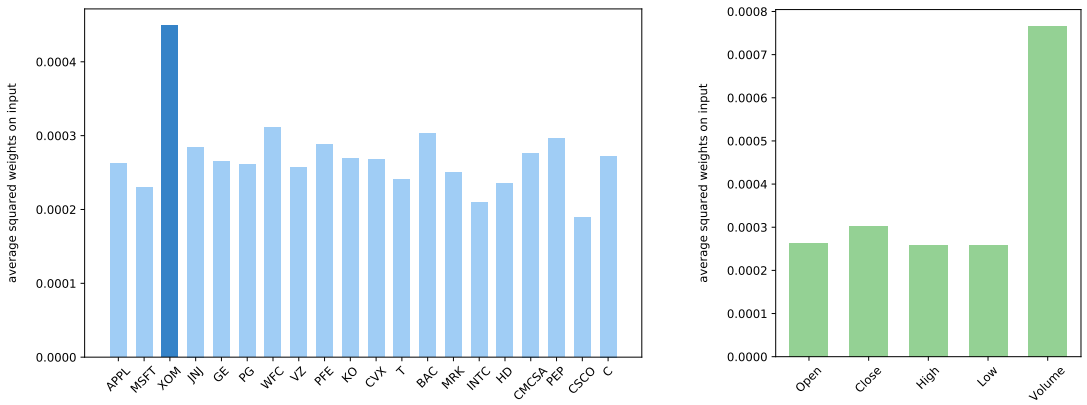


Figure 7: Examining the importance of inputs by looking at the learned filter weights in the convolutional layer. The network has been trained on XOM (rows 11-15). (a), (b), and (c) each show one channel of the convolutional layer of a trained network as a Hinton plot. Green squares indicate positive weights, red squares indicate negative weights. The size of the square corresponds to the absolute value.



(a) **(b)**

Figure 8: Examining the importance of inputs by plotting the average squared values of the trained weights in the convolutional layer. This network has been trained on predicting XOM closing price. Plot (a) compares different stocks while plot (b) compares different metrics.

in the first layer is then a measure of importance of the closing price as input for the network. In order to obtain meaningful results, the activation function would have to be set from ReLu to linear but using the trained weights from the original model. This approach is more complicated from a technical perspective and is therefore an open task for future work on this project.

However, figure 8 already shows the potential of weight visualisation. In this case, the model was trained for predicting the XOM next day closing price. The bar chart of figure 8a shows that indeed the weights referring to the XOM input values are the highest. This may seem unsurprising, but the network has no prior knowledge about which input data refers to the prediction task. Hence, in finance - where patterns are extremely hard to detect - it is a very good sign that the model seems to pick up which of the inputs belong to the stock it is predicting. Moreover, from this chart it can also be seen which other stocks are more and which are less important.

Figure 8b shows which metrics are particularly important for the network. 'Volume' should be ignored at this point due to its very different structure of the input data and resulting challenges in the normalisation. However, the figure shows that the closing price is the most relevant metric when predicting closing prices. Again,

this might seem unsurprising, but it shows that this kind of visualisation can be used to strategically select the ideal set of metrics in the future.

4.3 Performance over Time

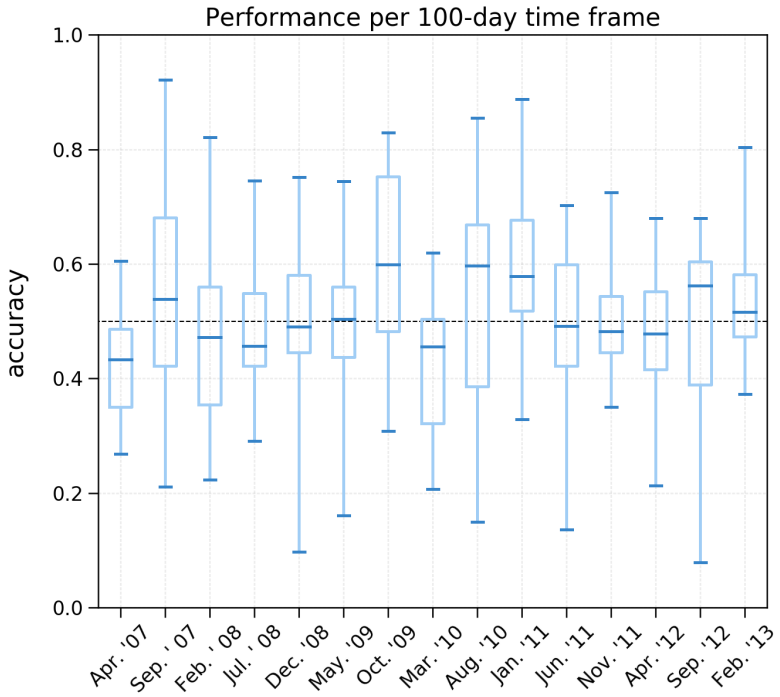


Figure 9: The performance of the algorithm over time for 20 different stocks visualised as a box plot. For each data point, the performance of 100 trading days was evaluated. The boxes indicate the 2nd and 3rd quartile while the whiskers denote the first and fourth quartiles. The labels on the time axes state the month in which the time frame begins. The dashed black line corresponds to a fictional benchmark of random guessing, i.e. an accuracy of 50%.

The test data set spans from 2007/04/24 to 2013/11/07 and the above stated results all refer to the performance over this entire time span. In this section, we will look at how the performance of the algorithm changes over time. To this end, the accuracies are evaluated for windows of 100 consecutive trading days. Figure 9 visualises this for all 20 stocks. Looking at the median performance, it seems that the algorithm performs close to random guessing in most time windows

but significantly above random in some time windows. It is worth investigating whether there is a possibility to assess *a priori* whether the market is currently in a phase of high predictability or not. An indication could for example be whether the algorithm currently produces a large amount of high certainty predictions or not.

One might assume that the performance of the algorithm should be better at the beginning of the test data set. While it is true that the beginning of the test data set is closer to the training data and therefore patterns could potentially be more similar, the algorithm does not have full awareness of the time dimension. More specifically, it does not know that the patterns learned between 2005 and 2007 might be especially relevant for predicting 2008 - of course, this is debatable, as markets in 2008 behaved much differently than the years before. However, it would be interesting to experiment with giving the network more information about the time dimension, for example by adding a recursive module to the architecture.

5 Conclusion

The suggested approach of applying a CNN to a high-dimensional input space and using only the very-high certainty predictions for predicting the stock market led to very interesting results with high potential. The architecture was optimized on the Apple stock, where it achieved an accuracy of 57.8%. Naively applying the same architecture to other stocks led to varying performances between 34.6% and 62.5% with a mean of 50.9%. The goal of future work is to either find methods to increase the overall performance on all stocks or to find a technique to detect beforehand whether the architecture works on a specific stock or not. Either of those two would enable investors to directly apply this algorithm on the stock market.

There are multiple ways in which this algorithm can be optimized. Major improvements can be expected when optimizing the input data. We developed a technique of visualising the amount of information the network retrieves from specific inputs. This can be directly used to make strategic decisions on the selection of stocks, technical indicators, or market indices like the S&P500 or the Dow Jones.

A major challenge of applying a neural network of this size to stock market data

is the limited size of training data sets. This challenge can be tackled with transfer learning [12]: In our case, the algorithm would be first trained on many stocks at once and the obtained parameters would then be used as initialisation values for the network when being trained to one specific stock.

Another interesting topic to look into is the incorporation of different time scales. In our case, the algorithm has the input data of the past 100 days. This enables the algorithm to learn by itself which time frames to look at. However, it might improve results if different time scales are directly incorporated by structuring the input data accordingly. One example would be to use more sparse or averaged data for the first 2 months and using hourly data for the last day. Non-equally spaced input data however might make it necessary to feed those different kinds of input to different convolutional filters.

References

[1] J. Patel, K. Kotecha, et al., *Predicting stock and stock price index movement using Trend Deterministic Data Preparation and machine learning techniques*. Expert Systems with Applications, Vol. 42, pp. 259–268, 2015.

[2] I. Goodfellow, Y. Bengio, A. Courville, *Deep Learning*. The MIT Press, Cambridge, 2016.

[3] N. Baba, M. KOZAKI, *An Intelligent Forecasting System of Stock Prices using Neural Networks*. IEEE international joint conference on neural networks, p. 371, 1992.

[4] T. Chenoweth, Z. Obradovic, *A multi-component nonlinear prediction system for the S&P 500 index*. Neurocomputing, Vol. 10(3), pp. 275-290, 1996.

[5] F. Fernandez-Rodriguez, C. Gonzalez-Martel, S. Sosvilla-Rivebo, *On the profitability of technical trading rules based on artificial neural networks: Evidence from the Madrid stock market*. Economics Letters, Vol. 69, pp. 89–94, 2000.

[6] R. Trippi and E. Turban, *Neural Networks in Finance and Investing: Using Artificial Intelligence to Improve Real-World Performance*. Probus, Chicago, 1993.

[7] I. Kaastra, M. Boyd, *Designing a neural network for forecasting financial and economic time series*. Neurocomputing, Vol. 10, pp. 215-236, 1996.

[8] S. Ghoshal, S. Roberts, *Reading the Tea Leaves: A Neural Network Perspective on Technical Trading*. Technical report, 2017.

[9] R. Akita, K. Uehara, et al., *Deep learning for stock prediction using numerical and textual information*. IEEE ICIS, Vol. 15, 16.

[10] J. Ticknor, *A Bayesian regularized artificial neural network for stock market forecasting*. Expert Systems with Applications, Vol. 40, pp. 5501–5506, 2013.

[11] K. Kim, *Financial time series forecasting using support vector machines*. Neurocomputing, Vol. 55, pp. 307–319, 2003.

[12] S. Pan, P. Yang, *A Survey on Transfer Learning*. IEEE Transactions on Knowledge and Data Engineering, Vol. 22 (10), pp. 1345 - 1359, 2003.

[13] W. Gilks, S. Richardson, D. Spiegelhalter, *Markov Chain Monte Carlo in Practise*. Chapman & Hull, London, 1996.