Algorithmic trading with a Convolutional Neural Network

Author: **Rory Murray**
Student ID: **17395111**

Supervisor: **James McDermott**

Code Source: https://github.com/Rorymurray37/FYP_CNN

BSc Computer Science and Information Technology National University of Ireland, Galway June 3, 2020

Table of Contents

List of Figures

1. Introduction

For my final project I sought to create a neural network capable of classifying profitable trades using historical stock pricing data. For many the stock market is viewed as an opportunity to make money. Investing in stocks can be a great way to grow your capital or can even result in a full time career. Now that markets are more accessible than ever, it brought with it lots of new attention to markets with people trading and investing in hopes of making profit. There are many different strategies that can be followed, however finding one that is proven to be successful can be very lucrative. For years and still to this day fundamental analysis and technical analysis have been the two main techniques when it comes to deciding which stocks to buy and when exactly to buy. However, in more recent times we've seen the success and progression of machine learning algorithms, with this they have been applied to stock market prediction and classification in hopes to find a new edge on the rest of the market. Models such as support vector machines (SVM's) [1] and random forests[2] have commonly been used for financial forecasting with some success, proving their ability to make accurate future predictions based on historic data. We have also seen deep learning models such as Recurrent Neural Networks (RNNs) [3] and Convolutional Neural Networks (CNNs) [4] applied to time series problems such as financial forecasting. CNN's are primarily known for their major success in image classification and natural language processing, however they can be adapted for financial forecasting. This can be done by formatting the 1D pricing data into 2D images which can then be used as inputs for the model. For my project I focused on implementing a CNN for the purpose of identifying profitable trades.

Rather than focusing on one specific type of trading strategy I chose to experiment with different types of trades and trade length. This was done to establish which types of trades the CNN works best with. When trading on a shorter timeframe there are a wide range of different trades that can be executed. A common one being the *hills and valleys* approach which revolves around identifying a local minimum and buying in hopes that it will rise from there. Others include buying whilst a stock is trending or even shorting a stock after the price has significantly risen. For this project I defined three different strategies I wanted the CNN to implement. One strategy was based around the hills and valleys approach and two strategies I developed myself.

To evaluate the model, I used a paper trade system which executes the model generated signals. This results in a final profit and loss for the time period tested on. As a baseline I used a Buy and Hold investment strategy as a comparison to my model. Buy and Hold is often used a comparison baseline for other trading strategies as the buy and hold strategy will always perform well when the asset being bought is performing well. The US stock market for the past ten years have been very bullish, this is reflected by the constant growth in the S&P 500 companies with tech stocks being at the forefront. Implementing a Buy and Hold strategy in market such as this is often a very profitable strategy. For this project I'm looking to investigate whether it is possible for my proposed CNN model to outperform the buy and hold.

Working in python I used jupyter notebooks to create the model with the help of several libraries including Pandas, Numpy and Keras. The inspiration for this project stemmed from both a deep interest in financial markets and machine learning, so I am grateful to be given the opportunity to work on such a subject for my final year project.

2. Literature Review

2.1 Financial Analysis

Financial analysis in this regard refers to analysis of companies or stocks in order to find out whether they are worthy of investing in or not. Both Technical and fundamental analysis fall under the umbrella of financial analysis. Fundamental analysis is a technique to determine a company's actual value by looking at a wide range of factors such as the company's revenue, debt, earnings per share, future outlook etc. There are additional qualitative factors that also have an influence on a company's valuation such as brand recognition/loyalty, patents, management and competitive advantage. All these different factors can make it challenging to put an accurate figure on a company's value, however educated estimates can be made. Technical analysis is the sole analysis of a company's share price over time. Technical analysts ignore a company's fundamentals and focus on factors such as pricing chart patterns, technical indicators and price trends. There is the key assumption that historic prices can be an indicator of future prices. Some traders and investors opt for one form of analysis or both. Both can have can their disadvantages when used on their own. For example, an investor may fundamentally analyse a company and deem them a high performing company and invest at given point in time, however the share price may currently be overvalued meaning buying at this point may result in losses. The flip side can also occur. A technical analyst may identify at a local minimum in the share price and buy a stock, at the same time news of poor performance of the company is released which causes the share price to drop further. It is widely agreed that fundamental analysis is more effective for long-term investing whereas technical is more effective for shorter term trades.

2.2Technical Analysis

Within the field of technical analysis a large number of indicators have been created over the years, mainly in the form of price functions. Relative Strength Index (RSI) is one such example. It is a momentum indicator which takes into account the average gain or loss in given window and assigns it a score from 0-100. If the RSI of a security is above 70 it is often interpreted as being over bought meaning a potential opportunity to sell / short as the security is overpriced. Using this indicator alone may prove successful in some scenarios and ineffective in others, such is the nature of financial markets. However, with the introduction of machine learning, algorithms can be created to identify patterns within indicators that occur at an optimal time to buy or sell. Technical indicators are extremely important when it comes to financial prediction as training models on the historical pricing data of a stock would not be sufficient data to learn from. The type of indicators selected may also have an impact on the performance and accuracy of the model. One possible way to optimise this is to apply a genetic algorithm to the selection process in order to select the most valuable indicators [10]. As well as that both fundamental and technical analysis can be combined and used as inputs to a machine learning algorithm as suggested by Lam [6]. This ultimately improved the performance of the network when compared with just training on technical indicators.

2.3 Trading and Investing

Trading refers to the execution of short -term buys or shorts in order to take advantage of short to medium term fluctuations in price. On the other hand, investing refers to buying a stock with a goal to hold it long term. this comes with support and belief in the company one is investing in. This is in contrast with a trader who may only be buying and selling a stock simply because it is volatile. Financial algorithms are almost always designed for trading rather than investing as breaking down the fundamentals of company is far more challenging for algorithms than analysing it technically.

This trend continues into financial machine learning models. They are primarily designed for trading as it plays into the strengths of the models these being its ability to learn from technical data. When it comes to trading they are many different strategies that exist that could be potentially learned by a machine learning model. Ideally a model such as neural network might be able find a new type of strategy that maximises profit. This may be possible by creating an advanced network that has no predefined strategy to learn and rather focuses on learning the most optimal method to make profitable trades. In most cases however machine learning models are trained using existing trading strategies with the goal to execute them as perfectly as possible.

2.4 Neural Networks

Within my project I decided to implement deep learning methodologies in an attempt to tackle this classification problem. Deep learning refers to artificial neural networks (ANN) that consists of multiple different layers that add features to the network with each layer improving overall performance [5]. This is one of the main advantages of deep learning that features of the data can be learnt without being specified beforehand. One of the most popular applications of deep learning in recent years has been image processing in autonomous vehicles. It has been a driving factor in the progress made in the autonomous vehicle industry. It has also been very effective at natural language processing which refers to a computer being able to understand and interpret the meaning of human language which can be very challenging for computers. Deep learning algorithms have proven capable of understanding the sentiment of financial news articles in order to determine how this news may affect the financial markets. Due to the large amount of data being generated nowadays it can be very time consuming to sift through it all manually so deep learning algorithms can be used to process all of it in an attempt to highlight potentially important pieces of information [6]. Before settling on implementing a CNN I had also researched using an RNN as a classifier for this problem. The RNN is an improvement on the original neural networks as it takes into account its previous inputs in the series of inputs rather than learning from one input on its own, meaning it has an internal memory which it uses to its advantage. This can be very helpful when dealing with time series prediction problems as shown by Connor et al. [7] where they created a RNN for prediction of electricity demand. Their RNN showed better performance on this dataset than that of a regular neural network. As a result RNN's have been regularly preferred to other deep learning algorithms when dealing with financial time series prediction[3].

2.5 Convolutional Neural Networks

Convolutional neural networks as mentioned previously have been very effective at image classification. They work by using a kernel to break down every image and learn features from each subsection of the image. One of the reasons for its success in image classification is down to the fact that within the image all neighbouring pixels are taken into account together rather than individually which means features of every pixel and related pixels are learned together. Layers of the CNN can consist of several convolutional layers which are directly preceded by pooling layers which reduces the outputs from the convolutional layer. For classification CNN's, flatten and fully connected dense layers are used to generate a final probability vector. This vector contains a value from 0 to 1 which can then be used to predict which class each input belongs to. In my case if the probability vector is greater than the threshold of 0.5 the associated input will be marked as a 'buy' and if it less than 0.5 it will be marked as a 'hold'. Dropout is another commonly used layer which is does as the name suggests and drops nodes from layers in order to account for overfitting. This results in a model which is very effective for feature extraction for images.

When it comes to time series prediction a CNN is not typically first choice due to the fact that the data is in the form of a series rather than images. However, it is still possible to apply the CNN to this problem by converting the time series into a set of 2D images. Omar and Ozbayoglu [4] showed this when they developed a CNN for stock prediction, training the model on 15x15 images made up of technical indicators. Once the time series is converted to images this problem becomes very suitable for CNN's to tackle. Another approach is to use a 1D CNN rather than a 2D model. A 1D CNN works quite well with time series data and does not require the creation of images. This was the approach I followed for my CNN with the goal of the model to classify profitable trading opportunities. Using technical indicators, I was able to expand upon the dataset and create multiple channels of inputs for the model rather than relying solely on the historic pricing data. The input for my model consisted of multiple timesteps of data, with each timestep containing pricing data and indicators.

Although the RNN may be more suited for time series prediction such as this the advantage of taking into account previous inputs in the series can also be replicated to an extent in a CNN through the design of the images, which I will discuss in detail at a later stage.

2.6 Machine learning and Financial Analysis

Financial time series forecasting is a highly researched field at the moment mainly due to the rise in popularity of data science techniques and machine learning in recent times. There are a vast number of models and algorithms being put forward some with very promising results [4]. ANN's, Genetic algorithms and hybrids models have been popular approaches to this task. Many models focus on attempting to correctly classify the optimal time to buy sell or hold, not to attempt to predict a stocks future price. Sezer et al [8] implemented an ANN based on trading signals generated from technical indicators. Their results indicated that by choosing the most appropriate indicators they could achieve a profit and loss comparable to that of the buy and hold approach. Chen et al [9] implemented a probabilistic neural network to forecast the direction of the Taiwan stock exchange index which proved to be effective also. An RNN was used by Kwon and Moon [10] whose input features were generated from a number of technical indicators. They also made use of a GA to optimise the model. Once again it was able to outperform the buy and hold strategy. In summary combining technical indicators with neural networks has proven to be successful in these certain cases above. Although it may be situational and may not perform in different scenarios, it does show that neural networks can be applied to time series problems with high accuracy.

3. Technical Review

3.1 Label

As the model is a classification CNN I need to create my own custom labels for this project. When working on defining the label for the model I took inspiration from Sezer and Ozbayoglu's [4] work on financial trading with CNN's. They created an algorithm for defining the *hills and valleys* approach to trading which works by identifying the min and max price points within a rolling window. Max points are labelled as 'sell' and min points labelled as 'buy' with the rest of the points being labelled as 'hold'. It also only labels the points if the min or max point in the window is equal to the middle point. This is done so labelled windows are consistent rather than the min or max points being in different indexes for every window. It also plays into the trading strategy ie buying a point which is both the min and middle of a window guarantees a profitable trade if it closed before the end of the window.

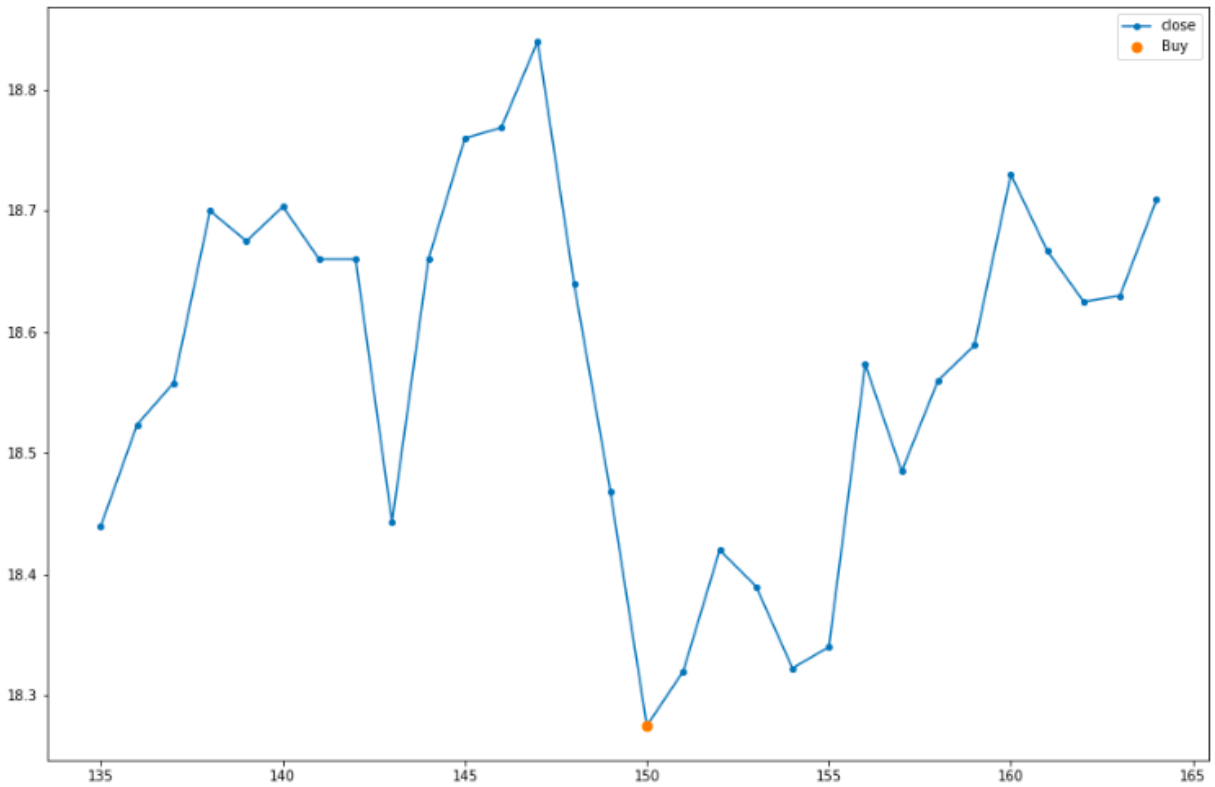

Figure 1 Sample 'buy' window of size 30

This is the ideal scenario for this strategy, the local minimum has been bought and the price has significantly increased after 15 time steps. In my case I chose to implement this technique for identifying min points for buying disregarding the selling aspect of it. This was a great initial label to work off as it is simple but effective given the model can learn it. When implementing a trading strategy for buying stocks, buying local minimum points is a very effective approach so this is a good baseline for comparison. This is one of the labels I implemented along with two others which I will discuss in the next chapter.

3.2 Input Features

When is comes to machine learning for stock market prediction technical indicators play a key role in the training dataset. They have been widely used when building a sufficient number of features

for training a model [4][8][11]. Technical indicators are important as they generate new additional information about a stock from its price alone such as whether it is overbought, trending or volatile. Overbought can be defined as when the majority of orders over a time period have been buys rather than sells. This can signify strength in a stock however it can also be a signal that there may be a sell off on the horizon as the stock returns to a natural balance of relatively equal buys and sells. The trend of a stock relates to its general price direction whether it is stagnant on the rise or falling. Volatility is another important concept in trading which described how much a stock's price is fluctuating over a time period. Stocks where large quantities of stock are traded every day would be described as volatile as it results in the share price changing quite frequently throughout the day. Although we may not see any clear relation between these attributes a stock may have, supplying the neural network with this additional data will most likely boost its performance as opposed to solely using the share price to learn from and generate predictions.

When it came to selection of the indicators I opted for some of most commonly used momentum indicators such as RSI, MACD, MOM and MA. I also included VAR and ATR which are two volatility indicators. Below are formulas for each indicator used.

- Relative Strength Index (RSI)

$$RSI = 100 - \frac{100}{1+(\frac{averagegain}{averageloss})}$$

- Moving Average (MA)

$$MA = \frac{A1 + A2 \ldots + An}{n}$$

A = average price in period n
n = number of time periods

- Moving Average Convergence Divergence (MACD)

$$MACD = 12\ period\ EMA - 26\ period\ EMA$$

EMA = exponential moving average

- Momentum (MOM)

$$MOM = \frac{Close\ price(i)}{Close\ price(i-n)}$$

n = number of time periods

- Variance (VAR)

$$\sigma^2 = \frac{\sum(x-\mu)^2}{N}$$

- Average True Range (ATR)

$$TR = \text{Max}[(\text{High} - \text{Low}), \text{Abs}(\text{High} - \text{Closing price}), \text{Abs}(\text{Low} - \text{Closing price})]$$

$$ATR = \left(\frac{1}{n}\right)\sum_{i=1}^{n} TR(i)$$

I calculated some of these indicators over several different time periods as there is no set time frame for these indicators that is deemed the best. This is similar to what was done by Sezer and Ozbayoglu[4], in their case they selected 15 indicators and calculated them over 15 different time frames. The idea of optimising the selection of technical indicators is something I explored but never implemented in the end. A great example of this is what Choudry and Garg[11] did in their paper. They developed a genetic algorithm for selecting a subset of optimal indicators from a large group. The fitness of each selection of indicators was the classification accuracy of their SVM model so it was a computationally intensive task given that they were working with 35 unique indicators.

3.3 Machine Learning techniques

When it came to building and training the model itself I followed the industries best practices as much as I could. These included creating distinct training and test sets, scaling the dataset, dealing with class imbalance and evaluating the model. Given that this was a binary classification problem of classifying whether to buy or hold the data was labelled as either a 1 being 'buy' or 0 being 'hold'. The labels I was working with defined quite specific trades which meant the majority of the price points in the dataset were labelled as 0. So within the dataset I was working with quite a large class imbalance which had a big impact on performance of the model. Techniques such as Synthetic Minority Over-sampling Technique (SMOTE) exist to solve this problem. SMOTE works by generating new synthetic examples of the minority class through use of a k nearest neighbours' algorithm. However I found I was able to combat the class imbalance using keras's class weights feature which allowed me to define a weight ratio for the label while it was training. For a number of stocks around 10% of the price points were labelled as 'buys' so accordingly I set the class weights as 10% for 'hold' and 90% for 'buy' examples. This significantly improved the model at classifying buys as previously it had been classifying almost every point in the test dataset as hold as it resulted in high accuracy of the model.

Since I was working with a binary classification problem, I used binary cross entropy as the loss function for the model. This works by creating score that penalises predictions from the model based on the distance from its actual class value. The model generated a prediction for every value in test dataset which was a probability vector between 0 and 1. If a prediction for a given price point was greater than 0.5 it was marked as a 'buy' signal. These 'buy' signals were then executed in a paper trade system, trading over the timeframe of the test dataset which resulted in a final profit & loss figure. This was another important evaluation metric of the model itself separate to loss and classification accuracy. So the goal of the model is to not just have high classification accuracy along with low loss but to also make sure the signals produced resulted in profitable trades.

3.4 CNN Architecture

The use of CNN's for stock market trading is still something that is quite novel. When it comes to machine learning models for algorithmic trading LTSM's, ANN's and linear regression models are more commonly preferred. In my case I intended to use a neural network of some kind for this task. After researching this field I took inspiration from those who had successfully implemented a CNN for this problem[4][12] and sought to create one myself. Although the majority of the work around CNN's that I had researched were 2D models I implemented a 1D model as it could work directly with the time series data at hand and did not require the creation of complex images from the raw data.

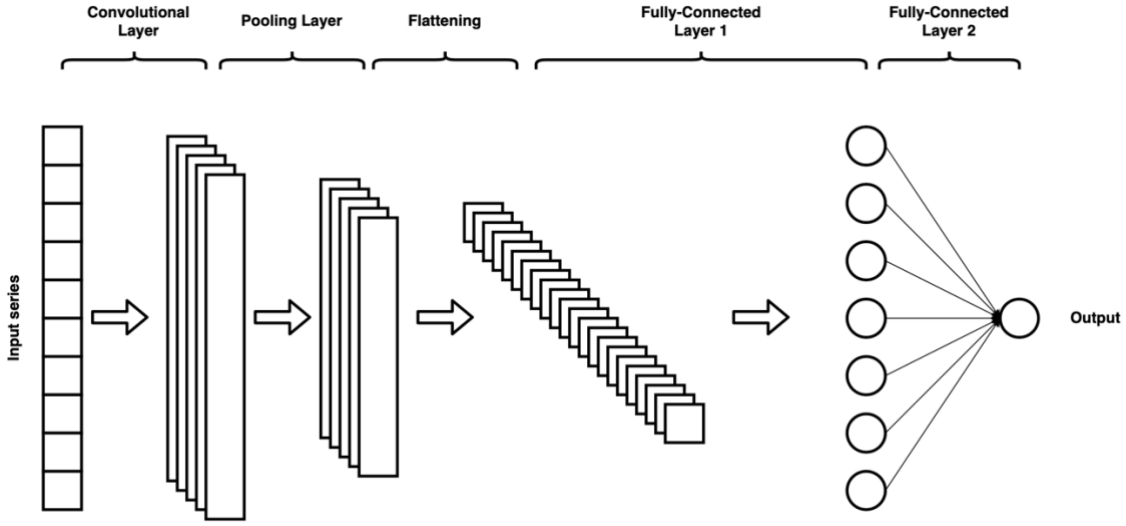

Figure 2. 1D CNN

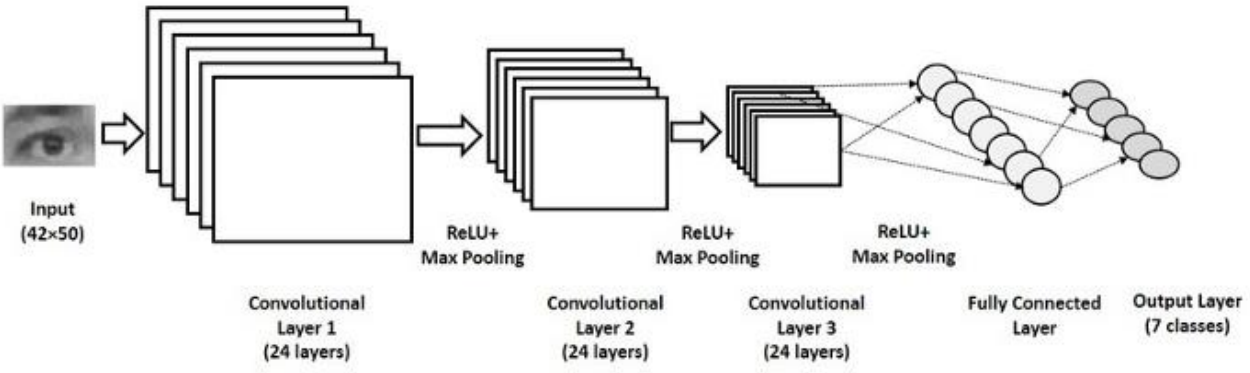

Figure 3. 2D CNN

From an architectural point of view both models follow the same format with the key difference being the type of data each network is designed for. Given that the 1D CNN can only work with 1D data this also means the kernel can only move in one direction also. This differs from a 2D CNN as the kernel moves in two directions in order to account for the extra dimension worth of data. For the convolutional layer the input size must be set so that it matches up with the dimensions of the input being used. In the case of a 2D model the input size would be the dimensions of the images being used. For a 1D model this would be the window size specified and the number of channels at each datapoint. In my case I used multiple channels for every timestep which included pricing data along with technical indicators. The above 1D CNN architecture is general format for such a network and is similar to what I implemented within my CNN.

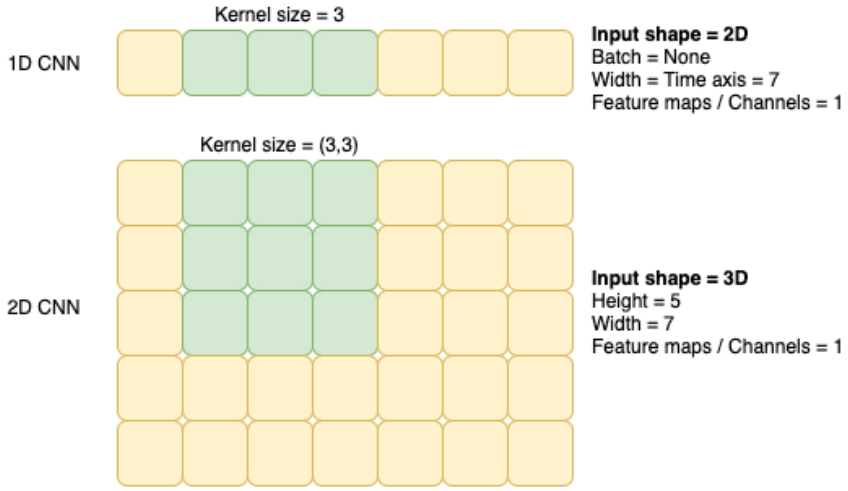

Figure 4. Example kernel from 1D and 2D CNN

From the above diagram we can see the differences between both kernels with the 1D kernel linearly sliding from left to right and the 2D kernel sliding from left to right with a shift down to the next row once it hits the edge of the input. The 1D CNN has a width or window size of 7, within my CNN the window size relates to the number of timesteps included in each input. The above 1D CNN also features only the one channel as opposed to 20 channels which is what was implemented within my CNN. This is the reason why CNN's can be effective at time series prediction as it has the ability to learn from multiple time steps at a time as opposed to learning from each timestep individually without taking previous timesteps into consideration.

The size of kernel specifies how many data points it learns from at each iteration through the window. It generally considered best practice to use a low odd number as a kernel size such as 3x3 or 5x5 for standard CNN's. Odd numbers are typically used for the kernel as using even sized kernel may lead to aliasing error where the initial image is downsized to a point where the image is considerably distorted. This error can occur from a combination of factors such as an even kernel size or if there a large number of convolutional and max pooling layers within the network. Within 2D CNN's the kernel is typically a small size in order to learn from close neighbouring pixels which are directly related. This helps generalize features from each image and not learn very specific traits that may not reoccur in other images. However, since I was working with a 1D CNN with timeseries data the kernel did not need to be constrained to smaller values as all neighbouring inputs were inherently related.

In order to get an understanding of how a 1D CNN would work from start to finish I researched several tutorials which I found to be quite useful. Keras provides a tutorial on using implementing a CNN with a multivariate time series. This helped me get a better understanding of the format of a CNN model and how to implement it within a python environment.

4. Method

4.1 Data Preparation

Initially sourcing enough data for this project was a challenge. Many of the websites that provide historic stock pricing data for free only provide it over large timeframes such as daily or even weekly time steps. It would be challenging and unrealistic to implement any kind of trading strategy using daily pricing data as the majority of trading strategies rely on the daily fluctuations in price in a given trading day to identify trading opportunities. Having one price value to represent a day worth of trading would simply not be a sufficient amount of data. Real world algorithmic trading systems would typically work with very high frequency data such as minutely data. Eventually I found a website called Finam [13] which offered historic pricing data over a wide range of frequencies. Given the choice I opted for half hourly pricing data. I chose this as I wanted the model to trade over a medium-term timeframe rather than trade very frequently over a short timeframe. This was something I was personally interested in, to see if an effective algorithmic trading system could be created that holds trades for a few hours or even days rather than one that buys and sells within minutes. However, this is only possible if the model is capable of identifying trends which last the duration of my proposed trades

The downloaded data came in the form of a csv which I acquired on the 8th of November later returning on the 28th of January for additional data. I gathered data on several stocks including Apple, Microsoft, Tesla, Delta, Google, BMW, Johnson & Johnson and Nvidia all over the same timeframe going from July 1st 2016 to December 31st 2020. This group of stocks consist primarily of high performing tech stocks with Delta, BMW and Johnson & Johnson being from different industries.

Each file contains the pricing data in the form of ‘open’ ‘high’ ‘low’ ‘close’ along with the volume, date and time. The open and close price in this case is the price at the start and end of the half hour interval, with the ‘high’ being the highest price recorded and ‘low’ being the lowest price recorded within the interval. Using jupyter notebook I read in the csv files and created dataframes for each stock using python’s pandas library. I then created a new datetime column which combined the date and time columns into one and set that column as the index for the dataframe. Doing this would allow me to split the dataframes based on the datetime rather than row number of dataframe which was useful. All unnecessary columns were then dropped from the dataframe. Since I was working with real financial data there were no missing values or duplicates, so the pricing data was now ready to be used to create additional features.

DateTime	Open	High	Low	Close	Vol
2016-07-01 10:00:00	46.75	47.3600	46.75	47.30	708938
2016-07-01 10:30:00	47.30	47.3600	46.95	46.98	398659
2016-07-01 11:00:00	46.99	46.9950	46.65	46.79	472746
2016-07-01 11:30:00	46.78	46.8299	46.61	46.66	325655
2016-07-01 12:00:00	46.66	46.7300	46.57	46.70	324933

Figure 5. Dataframe containing Nvidia data

4.2 Feature Creation

As mentioned previously in the Technical Review I used technical indicators to expand upon the base dataset. I used Ta-Lib for this task, a python library which has a wide range of functions to calculate specific indicators from the price data. In the end I calculated 15 indicators using multiple variations of the RSI, MOM, ATR and MA indicators. The additional variations of the indicators are generated by using different time frames, which results in different indicator values.

```
df['RSI_14'] = ta.RSI(df["Close"],timeperiod=14)
df['RSI_10'] = ta.RSI(df["Close"],timeperiod=10)
df['RSI_6'] = ta.RSI(df["Close"],timeperiod=6)
df['RSI_2'] = ta.RSI(df["Close"],timeperiod=2)
df['MOM_8'] = ta.MOM(df["Close"], timeperiod=8)
df['MOM_4'] = ta.MOM(df["Close"], timeperiod=4)
df['ATR_14'] = ta.ATR(df["High"], df["Low"], df["Close"], timeperiod=14)
df['ATR_10'] = ta.ATR(df["High"], df["Low"], df["Close"], timeperiod=10)
df['ATR_6'] = ta.ATR(df["High"], df["Low"], df["Close"], timeperiod=6)
df['VAR'] = ta.VAR(df["Close"], timeperiod=5, nbdev=1)
df['MA_10'] = ta.MA(df["Close"], timeperiod=10, matype=0)
df['MA_30'] = ta.MA(df["Close"], timeperiod=30, matype=0)
df['MA_50'] = ta.MA(df["Close"], timeperiod=50, matype=0)
df['MA_200'] = ta.MA(df["Close"], timeperiod=200, matype=0)
df['MACD'],macdsignal, macdhist = ta.MACD(df["Close"], fastperiod=12, slowperiod=26, signalperiod=9)
```

Figure 6. Code used to create set of indicators for each dataframe

I stuck with this set of indicators in this order for the entirety of the project. This is another aspect that could be experimented on, the choice of indicators and choice of time-period for each. Changing these may have an effect on the performance of the model. The order of the indicators is something that can have also have an effect on the model as the kernel in a CNN will learn from features that are neighbouring each other. Given more time this would be something I would've experimented with.

When I first began building the model, I had all the code located in the one jupyter notebook. As the project got larger, I began creating functions for specific tasks and importing them from other python files into the notebook. This helped tidy up the notebook and was also useful when it came to automating the model. The snippet of code above is located in the data_prep.py file which is used to prepare all the stock dataframes. Once the features had been created each timestep within the dataframes had 20 columns with 5 columns for pricing data and 15 for indicators.

4.3 Labels

As mentioned previously the label for this model is something I experimented with quite a bit. I worked with three labels named them standard label, trend label and bullish label. They all refer to different types of trading strategies which I wanted the model to attempt to learn and implement. All labels are calculated using the closing prices in each timestep in the dataframe. The labels are all designed to find optimal points to buy for a given strategy, they do not specify when to sell. I experimented with trade length as we will see later in the paper.

Standard Label

The standard label is based off the *hills and valleys* strategy mentioned in the technical review. It works by identifying the minimum point in a given window, if that minimum point is also the middle point in the window that price point is labelled as 1 which is a 'buy'. All remaining points are label as 0 which is 'hold'. A rolling window approach is used over the entire dataframe so every point in the dataframe is labelled.

Trend Label

This label works similarly to the standard label by selecting the minimum point in the window, however it is only selected as a ‘buy’ if 98 price points (which is equivalent to 7 trading days) forward the closing price of the stock has risen by a specified threshold such as 5% or more. In this case the minimum point does not also need to be the middle of the window. This label is designed to find a minimum point before a considerable positive uptrend. The threshold was made adjustable so a range of different thresholds could be tested. I created this label with a medium-term trading strategy in mind with the trades being held over several trading days as opposed to buying and selling within the same day.

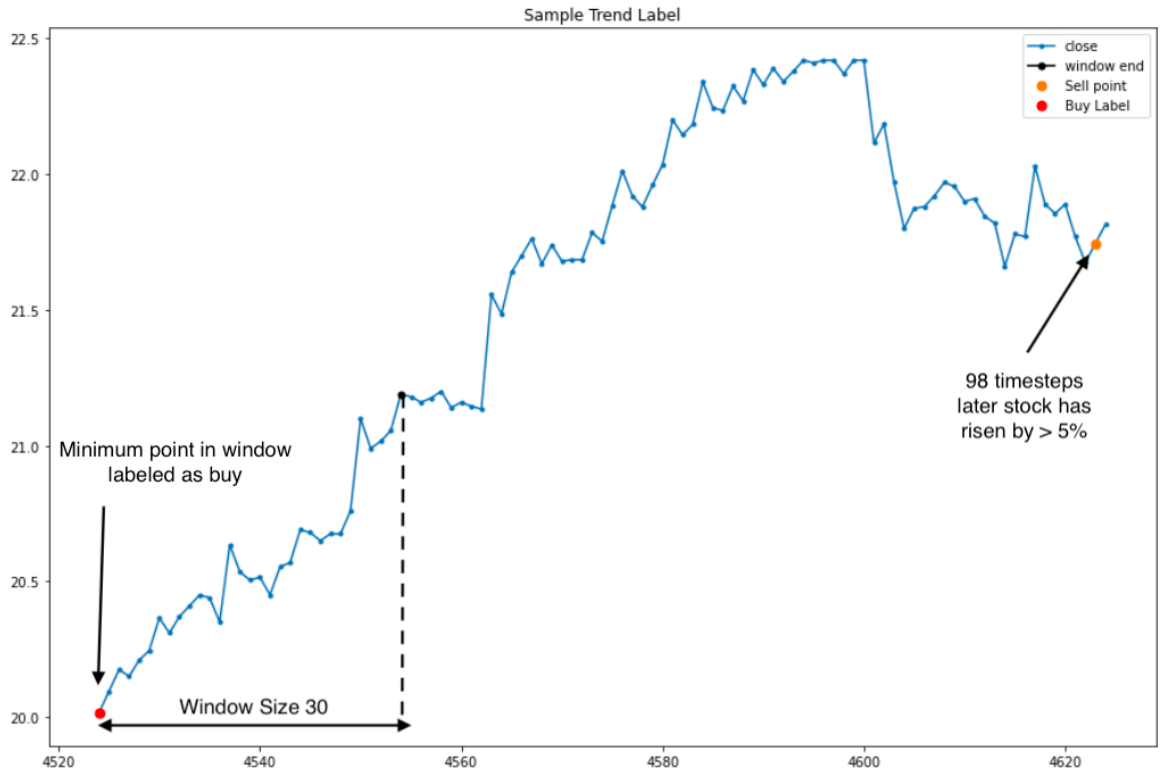

Figure 7. Example Trend label with a window of size 30 used

In the above we see how the label works. In this case the minimum point within this window of size 30 was the first point of the window. The algorithm will then check 98 timesteps ahead to the point in orange in order to calculate the change in price. Here the price has risen by more than 8% which is greater than our threshold of 5% so the point in red will be labelled as a ‘buy’.

Bullish Label

This label works by selecting the middle point in the window as a ‘buy’ if the start point of the window is equal to the minimum and the end point is equal to the maximum. This results in strategy that only buys a stock when its price is already rising. This differs from the other two labels as it is not looking to buy at minimum points. This type of strategy will only perform well on growth stocks hence the name *bullish* which is a term used in the financial world to express optimism that share prices will rise.

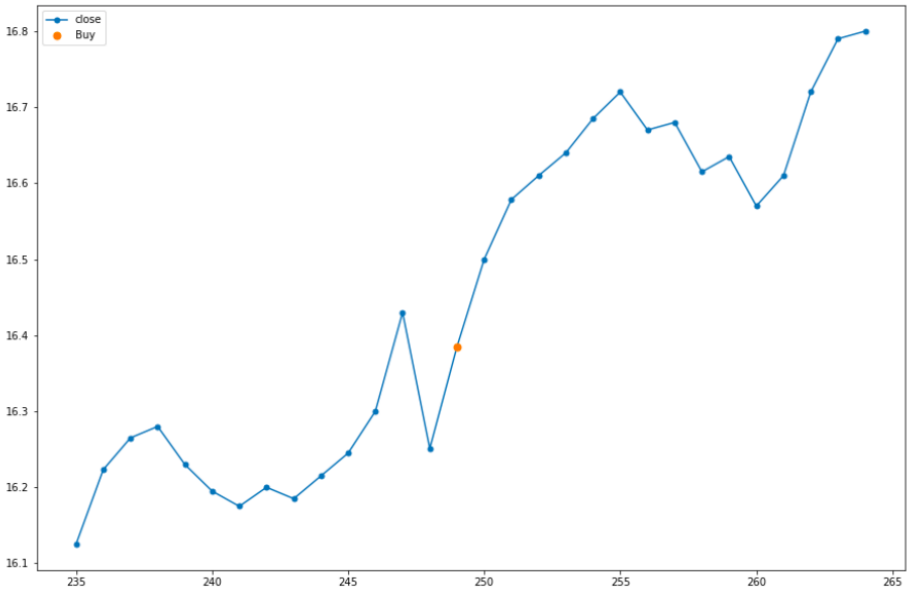

Figure 8. Sample ‘buy’ window of size 30 for the Bullish label

Figure 9. Example of both the bullish and trend label over a period of 800 timesteps

We can see how both labels differ from the above figure. The trend label labeling minimum points before the price rises whereas the bullish label is labeling points as the price is rising. You may also notice how there are far more buys for the trend label rather than the bullish label, this is because the constraint of being the middle index within the window.

4.4 Paper Trade System

Once I had all the price points for each stock labelled, I needed a system to execute the ‘buy’ signals generated by each label. This system works as follows, it iterates through the dataframe and executes buys if the label at a given timestep is equal to 1. It buys the stock at that point in time and holds the shares for a fixed length at which point it sells the stock. The system starts with a fixed balance in dollars, if the system attempts to buy shares but does not have a sufficient balance to do

so the shares will not be bought. This may happen if the system is already holding a large number of shares at one point in time. Once again when buying and selling shares at a given timestep the close price is used for calculations. The system has a starting balance of 100000 in dollars with each buy signal costing <= 10000 this means the system can only have 10 active trades at once given the balance has not dropped at this point.

When a buy order is executed as many shares as possible are bought from a fixed amount in dollars that fixed amount being 10000. This means the system is only buying whole shares as opposed to fractional shares. For example, if the system receives a buy order for Twitter at the price of 36.70 10000 dollars is divided by the share price. Since this does not result in a whole number the next lowest whole number of shares is bought which in this case is 272 shares which will cost 9982.40 dollars. The cost of shares is deducted from the balance and a position object is created which stores the entry price, number of shares bought and sell price. The sell price being a fixed amount of timesteps in the future e.g., the price 30 timesteps later. This was the approach I followed when it came to deciding when to sell for each trade. Every time the paper trade system was run an argument containing the trade length was passed in, this would determine how many timesteps to hold each trade for. This was another aspect of the project that I experimented with, testing various trade lengths. For a large part of the project the system was buying a fixed number of shares rather than a fixed amount in dollars. I eventually had to change this as it was not a fair system to use for all stocks as buying 10 shares of Twitter at $30 is not equal to 10 shares of Nvidia at $400.

Every position object that was created was also stored in list which contained all active positions. As the system was iterating through the time series it continually checked to see if the sell price had been reached for any of the active positions. If this was the case the sell function was run which closes the position by firstly multiplying the current share price by the number of shares originally bought for that position. This figure is added back to the balance and the position object is removed from the list and deleted. Once the loop had reached the end of the dataframe the profit & loss, number of trades executed, number of profitable trades and the average gain for each profitable trade were displayed.

```
Final balance :  160064.0
Final holdings :  0.0
Number of profitable trades  309 / 506
Average P&L for profitable trades :  5.152434960639023 %
P&L :  60064.0   Gain :  60.06399999999999 %
```

Figure 10. Sample output from paper trade system

4.4 Building Training & Test Datasets

With the stock dataframes now complete with features they were then split by date. I reserved all of the 2020 pricing data for testing and used from July 2016 up to 2020 as the training dataset. This worked out as a split of 23:77 for the test and training datasets respectively. Next, I scaled the data using sklearn's MinMaxScaler. The data was scaled between 0 and 1 with the scaler for both the training and test being fit using the training dataset. This is done so no additional information is learned from using a scaler fit on the test dataset such as the mean of the data or the standard deviation. This ensures all information learned comes from the training dataset.

Once I had a complete dataset and the label defined the next stage was to build the neural network. Having no prior experience with deep neural networks starting out was quite challenging. In order to get more familiar with the architecture and the process of building a network I followed a tutorial from tensorflow learn [14] which showed how to implement a number of different machine learning

models on the same weather dataset. This along with additional research of CNN's helped me get a good understanding of how I might approach building a network for my specific problem.

The first problem I faced was the format for the training and test data. Up until then I had been working with pandas dataframes to manage the data, however tensorflow does not work with pandas dataframes instead it works with numpy arrays. I had to transfer the data from the dataframes into arrays in order to input the data into the model. The input to the CNN is a Tensor of shape (x, y, z) where x is the number of time-steps, and y is the batch size and z is the number of channels. An example input would be (11215, 15, 20) where there are 11215 timesteps in the training dataframe a batch size of 15 timesteps and 20 features. The batch size or number of timesteps in each input for the model was something I had control over and could adjust to any size. However, after some testing I settled on having the input size as half the size of the label window used for simplicity. As an example, if I had labelled the data using a window of size 30 timesteps the input for the model would be an array with 15 timesteps worth of data. So, for each labelled price point within the data frame an array would be created containing the previous 14 timesteps leading up to and including that point. This would provide the model with sufficient data to learn from when given a labelled example of the type of trade it is looking to emulate.

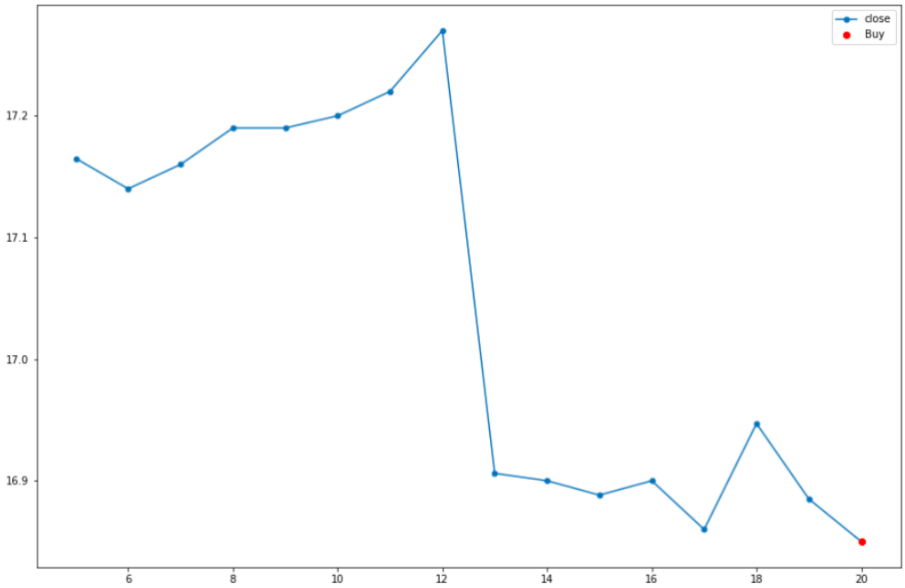

Figure 11. Example training input for the CNN

The network will receive the above price points in the form of an array along with a label set as 'buy' in this case. In order to create these input arrays for each row in the dataframe, I looped through each stock dataframe individually and added the newly created arrays of to one large array. Finally, I used numpy's stack function to join all the different stock arrays in to one 3-dimensional array which was now ready to be passed to the network as the training dataset.

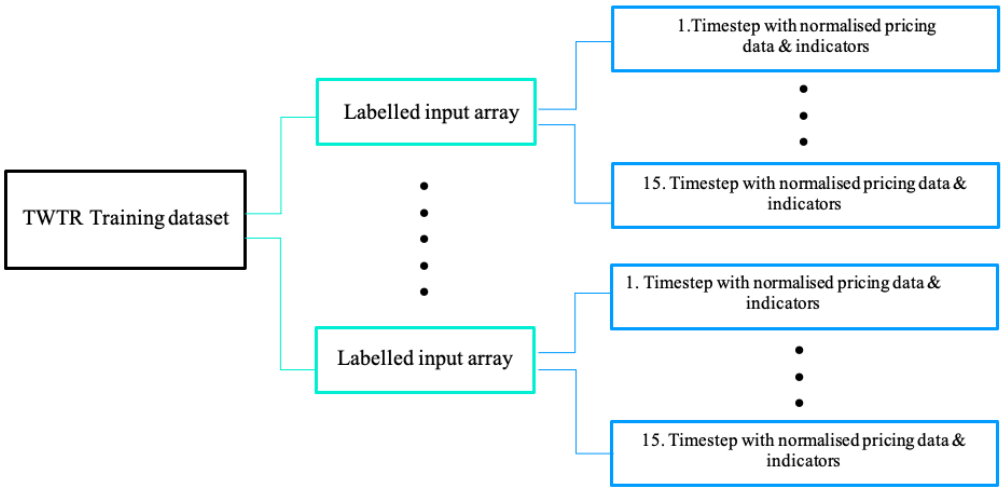

Figure 12. Format for a stock's training dataset where each input array contains 15 timesteps of data

An array with format above was created for every stock that was included within the training dataset. There was also a corresponding dataset containing the label for each input arrays with either a 1 or 0 for a 'buy' or 'hold'.

4.5 Network Architecture

When building the model, I used keras's library to implement the CNN itself. Keras's libraries provides functionality for a wide range machine learning models in python including convolutional neural networks. Importing the library allowed me to easily implement each individual layer of the CNN with a call to the library. On top of that it provides functionality to fit, evaluate and test the model. As mentioned earlier I implemented a 1D CNN which meant 1D convolutional and max pooling layers were used within the model. In the final version of the network I used two convolutional layers with two max pooling layers directly preceding. I set the filter size to 32 and 64 for the first and second convolutional layer respectively. I followed general best practice using 2 as the pool size for the max pooling layer which offered the best performance with higher values often resulting in aliasing errors. Within the first convolutional layer it was necessary to specify the input shape which was the dimensions of the input arrays I generated. Along with that padding of zero values was added to each convolutional layer to account for the kernel potentially extending over the boundaries of the input. Padding also preserves the border values and prevents against the input being reduced too small too quickly. A flatten layer was then added to convert the output into one dimension rather than two. Finally, an additional fully connected dense layer was added along with a dropout layer to account for overfitting.

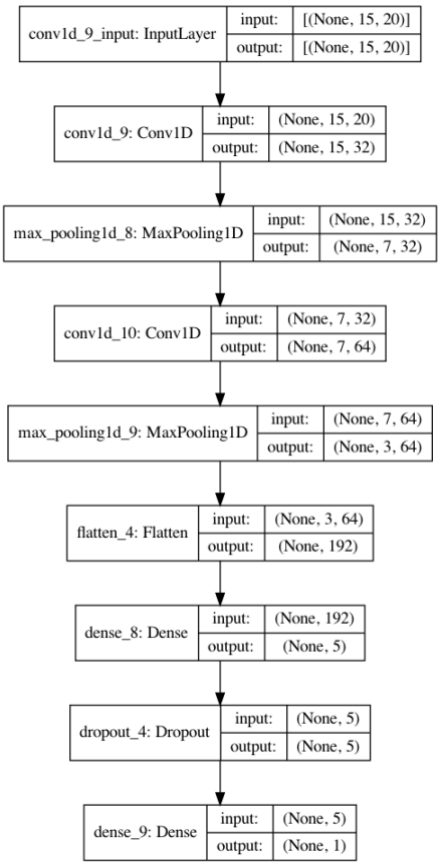

Figure 13. Final Network architecture used

4.6 Training

Each version of the model was trained for 30 epochs. A problem I quickly encountered when training the model was the imbalance within the data. Due to the labels I defined being quite specific there was often an imbalance with 95% of the values within the dataframes being labelled as hold and 5% being labelled as buy. The model simply predicted hold for a every price point within the test dataset resulting in high accuracy. In order to combat this, weights were introduced while fitting the model. The weights were set as 95:5 in favour of the buy class. This was done using the class_weights parameter in the fit method. This heavily prioritised the buy class when it came to training and prediction and vastly improved the performance of the network when it came to correctly identifying points to buy at. Although not all stocks had an exact data imbalance of 95:5 I found this value to offer the best performance, so it was fixed for the duration of the project.

4.7 Testing & Evaluation

I used keras's evaluate function to evaluate the model using the 2020 test dataset for each individual stock. Binary cross entropy was used as the loss function for the model given that it was a classification problem. Binary cross entropy penalises the predicted probability based on its distance from the actual value of the class. For each test run the accuracy and loss were recorded, with the goal keep the loss value low and the accuracy high. The paper trade system was then used to financially evaluate the predictions generated by the model.

After a test dataset has been evaluated by the above metrics predictions are then generated by the model. The predictions generated by the model came in the form of a probability vector from 0 to 1 to determine which class the given input belongs to. I used a threshold of 0.5 to split the classes. So,

if the network generated a prediction of 0.7 for an input it would be marked as a buy, any value less than 0.5 meant it was not a 'buy' so no action was taken. Once this was done for the entire test dataset the 'buy' signals were then executed in the paper trade system and the results were displayed. Upon each test of the model, the results from both the model's evaluation metrics and from the paper trade were noted. Despite this being a classification problem, it is also important to take into account the profits being generated by the models' signals. As having a model that generates predictions with high accuracy but results in poor profits is not of much use. Throughout the entirety of the project I tested many variations of the models with the model being trained on various combinations of the stocks. It was only possible to do such a level of testing with an automated model.

Another factor I had to consider was the performance of the stocks I was testing on over 2020. Although many stocks suffered at the start of the year due to pandemic the months preceding were very good for the stock market overall. Many stocks returned to pre pandemic levels or even broke new highs as the year went on. A number of the stocks I tested my model performed very well for the year which meant any kind of strategy that regularly bought the stock would have resulted in considerable gains. This was something I noticed my model was doing executing > 2000 buys over the year which would result in large gains. This was a false sense of success as the model was naively buying very frequently and getting lucky that the stocks it was buying were continuously rising. In order to combat this and test the model more rigorously I tested the model using stocks that did not perform very well over the course of 2020. I also altered the paper trade system such that it bought 10000 worth of every stock upon each buy signal resulting in less overall trades for the year but each trade carrying more weight.

4.8 Automation

When building the model I initially kept all the code and functions used within one python notebook. This size of the notebook eventually grew quite large and unorganised. In order to make the model more streamlined I moved all the functions related to data preparation and label generation to standard python files. I then imported the python modules to the model notebook which meant I could easily call each function with one line of code significantly reducing the size of the model notebook. Along with that in the early stages of development I was only testing the model on data from one stock alone, in order to prepare and label each stock dataframe it was necessary to have each of the processes in the form of functions. With this setup I was then able to loop through each of the stock dataframes preparing and labelling the dataframes individually. I then created one large function containing all the code for the model. This meant I could prepare all the stock data, train the model and test model by simply running one function. Parameters were added to this function which allowed me to test the model with various window sizes for the model along with testing various kernel sizes. To keep track of each run of the model the summary of the network were output to a text file along with training accuracy and loss. The model evaluation results along the results from paper trading the models' signals were also recorded in a text file.

5. Experiments & Results

The main experiment for this project was the performance of each different label I had created. Given that these are three differing trading strategies it was interesting to observe how they performed over each of the different stocks along with the model's ability to learn and implement the strategy. However, the overall goal of the model was to find the most profitable strategy which can adapt to different stocks and market conditions. All three of the labels are guaranteed to be profitable in any scenario if it can be implemented perfectly, so the performance of each trading strategy is dependent on the model's ability to learn the strategy. If the label is quite complex the model may struggle to learn it and implement effectively over the test datasets. The purpose of testing these various labels being to establish which strategy is the most profitable and can such a strategy implemented by a neural network outperform the buy and hold strategy.

Within my main experiment, the model was trained on Apple, Nvidia, Tesla, Microsoft, Google and Twitter. All of which are high performing tech stocks which each had considerable gains throughout 2020. The idea for this was to train the model on a group similar stocks in order to improve accuracy when testing on these types of stocks. I found this to be the case when the test results from NVDA were compared to a model trained on NVDA alone and a model trained on the above group of stocks. The model trained on the group of stocks had significantly better accuracy and loss when tested on NVDA when compared to the model trained on NVDA alone. I tested each label using three different window sizes. Experimenting with different window sizes for the label effects the size of the inputs that are received by the model. A window size of 30 results in inputs of size of 15x20 for the model whereas a window of size 100 results in inputs of size of 50x20. This additional information supplied may improve performance or hinder it due to the additional unrelated data.

On top of that I experimented with different trade lengths when paper trading the generated signals. The trade length was specified each time the paper trade system was run which determined how many timesteps each trade was held for. For my tests I opted to test a trade length of 100, 50 and 30 timesteps. The trade length of 100 plays into the trend label strategy which was to hold trades for a week which is approximately 100 timesteps. The other values were selected to provide a good variation in trade length. This style of selling may be considered naïve as it does not sell at a target profit or sell at a target loss it instead sells after a fixed amount of timesteps have elapsed. Not setting a target profit or loss does mean you could potentially receive higher profits but also higher losses. Overall, the success of the strategy is dependent on the entry position of the trade.

The buy and hold strategy was used as a baseline for comparison to the performance of the neural network. With the goal to see if a convolutional neural network such as this can outperform the buy and hold strategy in terms of profitability.

Buy and Hold Performance 2020

Stock	Profit & Loss
GOOGLE	28%
MICROSOFT	39%
NVIDIA	120%
TWITTER	67%

The above table lists the stocks I used for testing my model on. As we can see all four of the above stocks grew considerably over the course of 2020. The following model was trained on Google (GOOG), Nvidia (NVDA), Microsoft (MSFT), Twitter (TWTR), Tesla (TSLA) and Apple (AAPL) from July 2016 up to December 2019.

When analysing the profits from each different trade length of 100, 50 and 30 it was the trade length of 100 which resulted in the highest profit across all the different tests. Holding each trade for 100 timesteps resulted in less trades being executed due to the balance of the account running too low due to active trades. However, by holding the trade for longer more profit was being made on each trade.

Stocks	Label	Label Window	P&L 100 points	P&L 50 points	P&L 30 points
NVDA	Trend	30	37.40%	30.27%	15.89%
TWTR	Trend	30	34.45%	39.69%	28.20%
MSFT	Trend	30	16.59%	7.26%	4.62%
GOOG	Trend	30	4.43%	4.49%	3.64%
NVDA	Trend	50	120.71%	98.51%	67.72%
TWTR	Trend	50	69.65%	47.25%	44.81%
MSFT	Trend	50	20.09%	23.48%	20.31%
GOOG	Trend	50	15.40%	4.54%	12.52%
NVDA	Trend	100	34.54%	21.44%	28.13%
TWTR	Trend	100	40.71%	49.27%	66.49%
MSFT	Trend	100	1.48%	0.50%	6.78%
GOOG	Trend	100	17.87%	6.78%	6.11%

Figure 14. Comparison of P&L selling 100, 50 and 30 timesteps

5.1 Trend Label

Stocks	Label	Label Window	P&L 100 points	Profitable trades	No. of Trades
NVDA	Trend	30	37.40%	29	35
TWTR	Trend	30	34.45%	115	179
MSFT	Trend	30	16.59%	33	35
GOOG	Trend	30	4.43%	10	10
NVDA	Trend	50	120.71%	276	412
TWTR	Trend	50	69.65%	181	263
MSFT	Trend	50	20.09%	103	164
GOOG	Trend	50	15.40%	112	200
NVDA	Trend	100	34.54%	53	82
TWTR	Trend	100	40.71%	148	208
MSFT	Trend	100	1.48%	20	31
GOOG	Trend	100	17.87%	59	82

Figure 15. Results from the Trend label tested on the three different label windows

When testing the trend label, it was a window size of 50 along with a trade length of 100 which produced the best results. The P&L for each of stocks tested on using this variation of the model showed performance similar to that of the buy and hold. With the model outperforming the buy and hold with a profit of 69% when trading Twitter. This strategy revolved around identifying minimum points before considerable increases in price as such it performed very well when trading stocks such as Nvidia and Twitter which had considerable growth throughout the year.

Stocks	Label	Label Window	Test Accuracy	Test loss
NVDA	Trend	30	0.90	0.78
TWTR	Trend	30	0.70	0.65
MSFT	Trend	30	0.93	1.23
GOOG	Trend	30	0.94	0.37
NVDA	Trend	50	0.52	0.73
TWTR	Trend	50	0.52	0.91
MSFT	Trend	50	0.82	0.61
GOOG	Trend	50	0.75	0.57
NVDA	Trend	100	0.87	0.57
TWTR	Trend	100	0.74	0.61
MSFT	Trend	100	0.92	1.08
GOOG	Trend	100	0.91	0.44

Figure 16. Accuracy and loss from tests using the Trend label

When looking at the accuracy and loss values for each test we see that the model had strong accuracy but poor loss meaning a significant number of incorrect predications were made. Despite this the model still showed good profitability when trading. When looking specifically at the accuracy values when the window size was set to 50 we see a noticeable drop in accuracy of the model however both of those tests did result in substantial profits.

Predicted

Actual

	Hold	**Buy**
Hold	2268	652
Buy	199	120

Table 1. Confusion matrix of Trend Test

5.2 Standard Label

Stocks	Label	Label Window	P&L 100 points	Profitable trades	No. of Trades
NVDA	standard	30	104.98%	194	290
TWTR	standard	30	67.44%	164	244
MSFT	standard	30	47.11%	152	210
GOOG	standard	30	18.97%	143	234
NVDA	standard	50	42.19%	82	116
TWTR	standard	50	78.94%	137	187
MSFT	standard	50	26.54%	92	125
GOOG	standard	50	18.28%	90	159
NVDA	standard	100	42.55%	85	124
TWTR	standard	100	31.14%	66	96
MSFT	standard	100	24.26%	77	97
GOOG	standard	100	5.91%	62	112

Figure 17. Results from the Standard label tested on the three different label windows

Stocks	Label	Label Window	Test Accuracy	Test loss
NVDA	standard	30	0.89	0.24
TWTR	standard	30	0.88	0.17
MSFT	standard	30	0.91	0.79
GOOG	standard	30	0.88	0.22
NVDA	standard	50	0.95	0.08
TWTR	standard	50	0.93	0.11
MSFT	standard	50	0.95	0.14
GOOG	standard	50	0.93	0.11
NVDA	standard	100	0.95	0.07
TWTR	standard	100	0.95	0.07
MSFT	standard	100	0.96	0.16
GOOG	standard	100	0.95	0.07

Figure 18. Accuracy and loss from tests using the Standard label

Testing the standard label over the same parameters and stocks it also showed good profitability with more overall consistency then that of the trend label. The best performance was reached when the window size was set to 30 for this label. The P&L for all four of the stocks when window size was 30 were quite good with a profit of 47% when trading Microsoft. This is significant outperformance of the buy and hold of 39% in this case. The model was also a lot more successful learning this label as opposed to the trend label as both the accuracy and loss values show improvements. Both the accuracy and loss values improved as the window size increased however this did not result in an improvement in profits.

		Predicted	
		Hold	**Buy**
Actual	**Hold**	3025	176
	Buy	16	22

Table 2. Confusion matrix from test with standard label

5.3 Bullish Label

Stocks	Label	Label Window	P&L 100 points	Profitable trades	No. of Trades
NVDA	bullish	30	66.31%	180	246
TWTR	bullish	30	57.53%	198	292
MSFT	bullish	30	24.14%	118	172
GOOG	bullish	30	12.74%	153	252
NVDA	bullish	50	30.74%	54	82
TWTR	bullish	50	22.54%	123	197
MSFT	bullish	50	10.60%	53	75
GOOG	bullish	50	11.72%	101	160
NVDA	bullish	100	32.96%	37	45
TWTR	bullish	100	-22.92%	77	133
MSFT	bullish	100	2.79%	14	18
GOOG	bullish	100	10.18%	50	72

Figure 19. Results from the Bullish label tested on the three different label windows

Stocks	Label	Label Window	Test Accuracy	Test loss
NVDA	bullish	30	0.87	0.23
TWTR	bullish	30	0.79	0.27
MSFT	bullish	30	0.91	0.22
GOOG	bullish	30	0.84	0.21
NVDA	bullish	50	0.96	0.08
TWTR	bullish	50	0.90	0.13
MSFT	bullish	50	0.96	0.12
GOOG	bullish	50	0.92	0.11
NVDA	bullish	100	0.98	0.05
TWTR	bullish	100	0.92	0.10
MSFT	bullish	100	0.99	0.14
GOOG	bullish	100	0.97	0.05

Figure 20. Accuracy and loss from tests using the Bullish label

Once again it was a window size of 30 which generated the best profits when using the bullish label. Profits were slightly less than that of Standard label and failed to outperform the buy and hold across any of the stocks tested on. The model showed high accuracy and low loss when working with this label.

When it came to learning each of the defined strategies the model struggled the most with the Trend label, this is most likely down to fact that this label uses information which is not contained within the actual window itself (checking a week forward to see if the stock has risen) to determine whether a given point is a buy or not. The two other labels determine whether a point is a buy or not based entirely on the data within the window. This resulted in the model being able to implement both strategies with higher accuracy.

When analysing the profits from the Bullish label we see that they did not reach as high a level as the other two labels despite the high accuracy of the model. This suggests that the underlying strategy of buying whilst a stock is rising is not as effective as looking to buy at local minimum points. Both the Trend label and Standard label showed profits close to or better than that of the buy and hold over 2020 showing that the suggested CNN can compete with the buy and hold.

Optimal parameters for the CNN

Label: standard
Window size: 30
Trade length: 100

P&L 2020

	Buy & Hold	**CNN**
GOOGLE	28%	18%
MICROSOFT	39%	47%
NVIDIA	120%	104%
Twitter	67%	67%

5.4 Additional Tests

The model was also tested on Delta airline a stock which had a poor performance throughout 2020 due to the pandemic. The buy and hold resulted in -31% loss over 2020. The model trained on Delta alone and tested solely on Delta. This was done to see how model performs when then underlying stock has lost value throughout the year.

Label	P&L	Accuracy	Loss
Standard	-19%	0.93	0.30
Trend	-8%	0.90	2.97
Bullish	-15%	0.96	0.45

Table 3. Results from testing the model on delta.

As we can see all tests ran a loss but not quite as heavy as the buy and hold of -31%. When looking at the number of trades executed all tests executed less than 50 trades highlighting the lack of buying opportunities during this period. The small number of trades might also be down to the highly irregular price movement which the model was not accustomed to so it may have generated a large number of 'hold' predictions as it wasn't sure of many 'buys'.

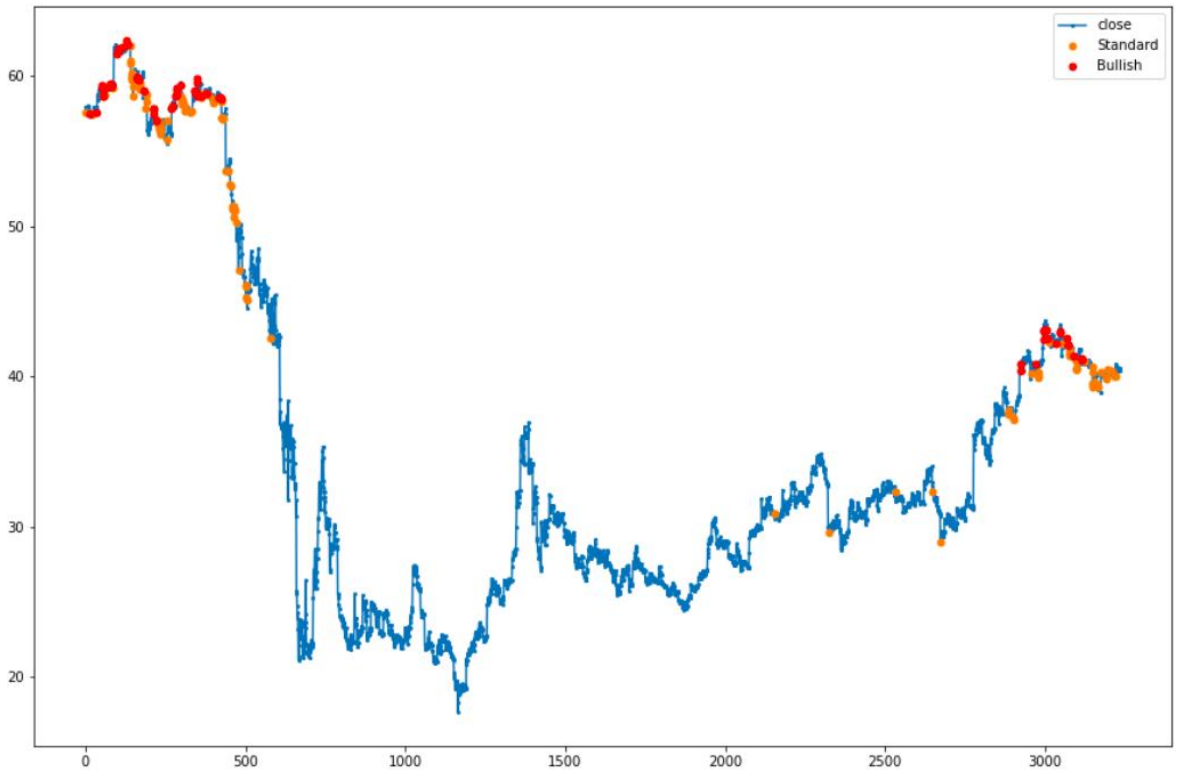

Figure 21. 2020 chart of Delta's share price with highlighted buys for the standard and bullish label

From the above graph we can clearly see the quite irregular price movement throughout 2020. With a very steep and prolonged sell off at the start of the year followed by high volatility for the remainder of the year. We can see the standard label identify local minimum points as the price is dropping resulting in large losses. It is worth noting that a selloff such as the one in March 2020 due to the pandemic uncertainty is highly irregular and very difficult to predict. In a situation such as this it is wise not to trade at all given the high volatility can pose lots of risks.

6. Conclusion

When starting out with this project the goal was to develop a model capable of making profitable trades within the stock market. As we've seen the model has done this with a performance on par with the performance of the buy and hold strategy. Each of the labels which defined specific trading strategies were all implemented to a certain degree of success by the network. Showing its ability to adapt and learn different strategies. A wide range of different trading strategies could be defined and tested using this model not just the ones I had used. Given more time additional tests could've been performed. Changing the threshold for classifying predictions as buys or holds could be changed from 0.5 to a higher value which may provide more accurate trades. Hyperparameter tuning could also be performed on the model which may boost performance. Also experimenting with different technical indicators and indicator combinations in the feature set may have an effect on the model's ability to learn the defined strategies. Overall, the suggested convolutional neural network has shown its ability to perform and generate substantial profits when applied to the problem of trading stocks a typically challenging task.

6.1 Limitations

If such a model was to be implemented in a live trading situation some additional factors would have to be accounted which were not addressed within this project. This model revolves around executing trades however the majority of brokers charge commission on each trade so this would have a significant effect on profit. Throughout this experiment trades were executed without subtracting a commission fee so in that sense this experiment is not entirely realistic as overall profits may be affected by as much as 5% when paying commission on each trade.

In order to set up the model to trade in real-time an api would be required to regularly retrieve live stock prices. These could then be formatted and input into the model which would then generate a prediction on whether to buy or not. However, prior to being used to trade in live situation more rigorous testing would be required over different years as the model may be able to trade successfully over 2020 but fail to do so at this point in time.

References

[1]Henrique, Sobreiro, Kimura (2018) Stock price prediction using support vector regression on daily and up to the minute prices. The Journal of Finance and Data Science, Volume 4, Issue 3, Pages 183-201, URL: https://www.sciencedirect.com/science/article/pii/S2405918818300060#bib32

[2] P. Chatzis, Siakoulis, Petropoulos, Stavroulakis, Vlachogiannakis, (2018) Forecasting stock market crisis events using deep and statistical machine learning techniques. Expert Systems with Applications, Volume 112, Pages 353-371, URL: https://www.sciencedirect.com/science/article/pii/S0957417418303798

[3]Xindi Cai, Nian Zhang, Ganesh K. Venayagamoorthy, Donald C. Wunsch (2007) Time series prediction with recurrent neural networks trained by a hybrid PSO–EA algorithm. Neurocomputing, Volume 70, Issues 13–15, Pages 2342-2353, URL: https://www.sciencedirect.com/science/article/pii/S0925231207000380

[4]Sezer, Ozbayoglu, Algorithmic financial trading with deep convolutional neural networks: Time series to image conversion approach (2018) Applied Soft Computing, Volume 70, Pages 525-538, URL: https://www.sciencedirect.com/science/article/pii/S1568494618302151

[5] LeCun, Bengio, Hinton (2015) Deep learning. Nature Volume 521, Pages 436–444, URL: https://doi.org/10.1038/nature14539

[6] Lam (2003) Neural network techniques for financial performance prediction: integrating fundamental and technical analysis. Decision Support Systems, Volume 37, Issue 4, Pages 567-581, URL: http://www.sciencedirect.com/science/article/pii/S0167923603000885

[7] Connor, Martin, Atlas (1994) Recurrent neural networks and robust time series prediction. IEEE Transactions on Neural Networks, vol. 5, no. 2, pp. 240-254, URL: doi: 10.1109/72.279188.

[8] Sezer, Ozbayoglu, Dogdu (2017) An Artificial Neural Network-based Stock Trading System Using Technical Analysis and Big Data Framework. Proceedings of the SouthEast Conference, Pages 223–226, URL: https://dl.acm.org/doi/pdf/10.1145/3077286.3077294

[9] Chen, Leung, Daouk (2002) Application of neural networks to an emerging financial market: forecasting and trading the Taiwan Stock Index. Computers & Operations Research, Volume 30, Issue 6, Pages 901-923, URL: http://www.sciencedirect.com/science/article/pii/S0305054802000370

[10] Y. Kwon and B. Moon (2007) A Hybrid Neurogenetic Approach for Stock Forecasting. Transactions on Neural Networks, vol. 18, no. 3, pp. 851-864, URL: doi: 10.1109/TNN.2007.891629.

[11] Choudhry, Garg (2008) A hybrid machine learning system for stock market forecasting. World Academy of Science, Engineering and Technology, Volume 39, no. 3, Pages 315-318, URL: https://www.researchgate.net/profile/Kumkum-Garg/publication/238747905_A_Hybrid_Machine_Learning_System_for_Stock_Market_Forecasting/links/00b7d53b4cfc215d81000000/A-Hybrid-Machine-Learning-System-for-Stock-Market-Forecasting.pdf

[12] Gudelek, Boluk and Ozbayoglu (2017) A deep learning based stock trading model with 2-D CNN trend detection. IEEE Symposium Series on Computational Intelligence (SSCI), Pages 1-8, URL: https://ieeexplore.ieee.org/abstract/document/8285188

[13] Finam (2020) URL: https://www.finam.ru/

[14] Tensorflow (2020) URL: https://www.tensorflow.org/tutorials/structured_data/time_series#convolution_neural_network