MATH70116 Deep Learning Assessed Coursework

Dominic Rushton CID: 06019954
Thomas Pink CID: 06003339
Charles Favell CID: 01851680

December 5, 2024

Contents

1 Building and Training a Binary Classifier

1.1 Notes on Methodology

In this report we detail the steps taken to build and train a binary classifier model, using deep learning to predict high-frequency price changes of two US stocks. We are provided with two sets of data; the first set is labeled and so is used to train our model, the second set is unlabeled and so we use our trained model to predict this data set's missing labels.

The first dataset, which will subsequently be referred to as `Data_A`, is structured as follows: It is a 200,000 by 22 array containing price data for two unknown stocks. These data points have been randomly drawn from a larger data set for both stocks, covering the period 1 August 2022 – 1 November 2024. Therefore the data points can be treated as 100,000 independent samples from each stock and as a result, no time series structure can be recovered. In the first column of `Data_A` is the label, the stock's midprice change direction, coded 0 to indicate a decrease in midprice or 1 to indicate an increase in midprice, where midprice is defined as midprice $= \frac{\text{bid price}+\text{ask price}}{2}$. Columns 2-22 represent the features, containing bid and ask prices and volumes from the limit order book as well as the five previous midprice changes.

The second dataset, which will subsequently be referred to as `Data_B`, is structured as follows: It is a 20,000 by 21 array with a further 20,000 samples (drawn similarly as those in `Data_A`) but with labels omitted.

Our approach begins by undertaking some feature engineering in an attempt to further understand and identify any patterns in `Data_A`. Following this, we split `Data_A` into training and validation sets and use hyper parameter optimisation to train the model attempting to maximise our accuracy when predicting labels for the validation set. This report is submitted along with a set of predictions to be tested with `Data_B` as well as our Python code which is included in the report's appendix.

1.2 Feature Engineering and Data Analysis

Before deciding on a machine learning model, we undertake some data preprocessing, in order to identify possible useful features that we may intuitively understand but the model may not capture.

1.2.1 Mean Price

We begin by considering the mean price of the data, in both the labelled and unlabelled data sets.
From analysis of the mean price alone in Figure 1, there doesn't appear to be any significant insight that we can gain. However it does suggest that the data was sampled at specific times, or perhaps that the stocks have very different characteristics, as there is a clear separation in the mean prices into groups within the plot.

1.2.2 Distribution of Price and Skew

We now consider at each point, the distribution of the prices of the stock, in essence treating each bid/call ask price, along with their volumes, as some discrete distribution of the price

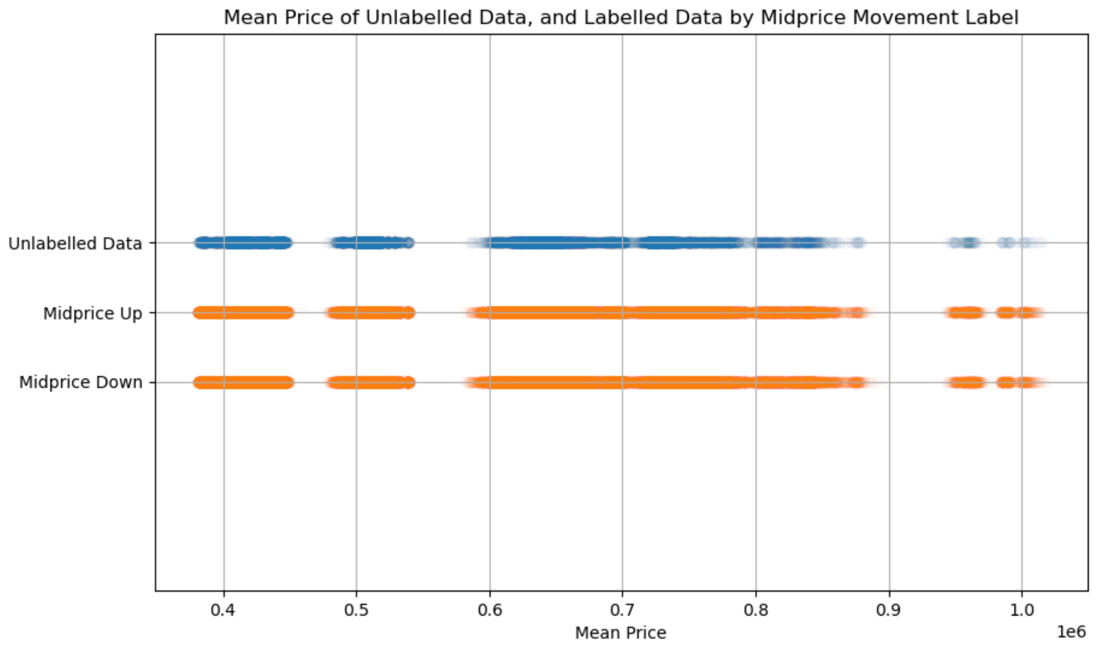

Figure 1: Mean of Bid/Ask Prices, of each data point, split by label.

random variable.

The idea behind this being that we hope if the price distribution is skewed in a certain direction, then the price will be more likely to move in that direction. An example is shown in Figure 2

We can then include statistics of the distribution, such as mean, variance, skew and kurtosis, to determine if they have any meaningful impact on the results.

As a useful substep in our calculation we also compute statistics analogous to the mean, variance, skew and kurtosis for the distributions within each individual order level as well. For example for the 'variance' at level 2, denoting the overall mean as $\overline{L}$:

$$Var(L_2) = (L_2^{bidprice} - \overline{L})^2 L_2^{bidvolume} + (L_2^{askprice} - \overline{L})^2 L_2^{askvolume} \tag{1}$$

Of these statistics, none appear to separate the two data sets, except for the skew statistic. Of the various 'skew' like statistics we computed the only item that appears to matter is the level 1 order book 'skew'. Specifically, we can see a clear separation in Figure 3 between the labels based on this statistic. Therefore we investigate this further.
Given skew of just a binary random variable will just indicate whether which of the two options had more results, we come up with a simpler statistic of the normalised difference between the level 1 bid and ask order books. (1, representing only bids, -1 representing only asks). Explicitly we compute this as:

$$L_1^{split} = \frac{L_1^{bidvolume} - L_1^{askvolume}}{\max(L_1^{bidvolume}, L_1^{askvolume})} \tag{2}$$

Predicting that the next midprice will move up whenever this statistic is greater than 0, and

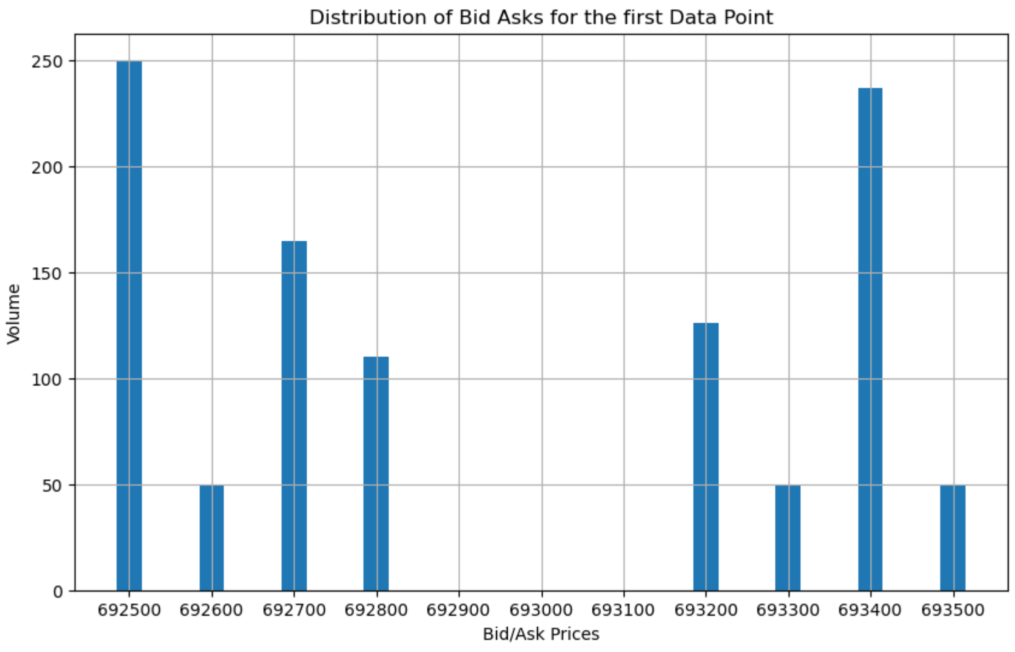

Figure 2: Distribution of the bid/ask prices of the first data point

down otherwise, gives accuracy of 68%.

Some intuition on why this may be so predictive, is that the level 1 order books are the closest to each other and therefore represent the instantaneous demand for the stock. If lots of people want to sell at a specific price, more than buy, then there is a mismatch in demand. In order for sellers to be able to win out in the bidding auction and capture the limited number of buyers, they must lower their offers, which causes the midprice to fall in the next movement.

1.2.3 Identifying Differences in Entry Behaviour

Since we now have such a predictive statistic, we will apply it to the data split by mean that we noted in Figure 1

Mean Range	Accuracy of L_1^{split}
$\overline{L} > 900000$	63.4 %
$795000 < \overline{L} \leq 900000$	63.4 %
$707500 < \overline{L} \leq 795000$	61.5 %
$550000 < \overline{L} \leq 707500$	62.4 %
$465000 < \overline{L} \leq 550000$	70.9 %
$\overline{L} \leq 465000$	77.0 %

Table 1: Table Showing Accuracy of Level 1 Order Volume Split test statistic at predicting price movements

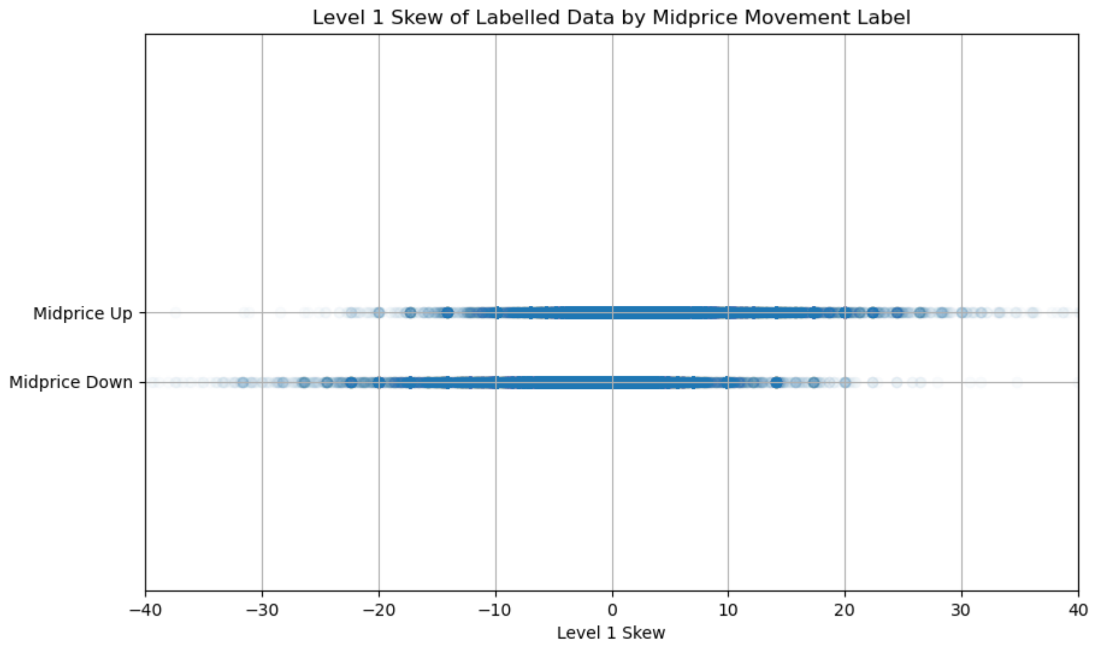

Figure 3: Level 1 'Skew' Statistic shows clearly different behaviour between the two labels

We actually see remarkable grouping of the value of this statistic depending on the mean of the bid and ask data within Table 1. If the mean is above 550,000, then we get around 63 % predictive accuracy with the statistic. For those items with means below 550,000 we get around 74 % predictive accuracy.

This suggests that the two groups behave very differently. Given that we know there are two stocks that have been sampled in the data set, it suggests that we are able to identify which stock is which based on their mean.

Doing so, we see actually that those items with a mean of above 550,000 correspond exactly to the first 100,000 points in the labelled data set, and those with mean below 550,000 the next 100,000. If the data was sampled one stock at a time, this would again be consistent with our analysis. We are further reassured knowing that there are 100,000 samples of each stock.

We therefore identify a new statistic, the stock label, based on the mean of the stock as described above, which we assign 1 for the first stock, and 0 for the second.

1.2.4 Previous Midprice Movements

In all our discussion so far, we have as yet not performed any analysis of the 5 previous midprice changes.

Initially we consider whether they may be some sort of mean reversion effect. For example, if the stock went up 5 times in a row, is it more likely to come down the next time? This did not appear to have very great predictive power, predicting correctly in only 55% of cases.

Similarly, we noted for a couple of items that the midprice movement appeared to be alternating. Up down up down etc. Under the assumption that this would continue, we could make a prediction based on an alternating sum say. But again the predictive power here was only around 55%. The intuition being that the price was fluctuating around some true mean.

Rather than trying to guess, we therefore just sum up over the labelled dataset based on each pattern of price movements in order to determine what the liklihood of the next flip was. We obtain Table 2 of results.

This shows us that alternating is actually quite highly predictive as suggested. In approximately 75% of cases it will continue to alternate midprice movement directions.

Other patterns appear to be less predictive. Indeed there does not appear to be a strong mean reversion effect. Predicting the labels based on the pattern, with a prediction of up if it went up more often than not in the data set, gives a predictive accuracy of just over 60%, which is a bit better than a coin flip!

However we can be fairly confident in the predictions. If it were just random chance, then reversing the movements in pattern may not necessarily result in the the same frequency in the opposite direction. However, when we add the opposite patterns, we do in fact obtain approximately 1 in each case, suggesting that these patterns are fairly robust.

For example, the chances for 01011 and chances for 10100 are 60.8 % and 39.8 %, which sums to 100.6 % $\approx$ 1.

1.2.5 Principal Component Analysis

We did not consider principal component analysis and cleaning of data by reconstruction through large eigenvalues, as our understanding is that this destroys noise in the dataset, which for deep learning is actually desireable to prevent overfitting.

1.2.6 Potential Normalising Methods

Finally we have to normalise the data, including all of the statistics we developed.

Initially we consider whether normalising along rows as much as possible was better. The idea in doing so is that we want to ensure that the structure of the distribution of volume and prices is preserved within each row, as this would appear to be the basis upon which a prediction for an individual data-point is made.

More explicity we normalise the prices via the mean and variance we computed for each row.

$$\frac{(L - \overline{L})}{\sqrt{Var(L)}} \tag{3}$$

And then normalise the volumes by dividing them by the total volume (of all bids and asks up to level 4).

Midprice Movement Pattern	**Frequency of Midprice Increasing**
00000	41.8 %
00001	47.0 %
00010	46.9 %
00011	50.8 %
00100	48.3 %
00101	52.2 %
00110	59.8 %
00111	51.4 %
01000	55.1 %
01001	62.1 %
01010	76.0 %
01011	60.8 %
01100	49.5 %
01101	55.8 %
01110	50.7 %
01111	46.9 %
10000	50.2 %
10001	46.9 %
10100	39.8 %
10101	24.6 %
10110	38.3 %
10111	44.0 %
11000	47.3 %
11001	40.4 %
11010	48.6 %
11011	50.0 %
11100	48.9 %
11101	51.1 %
11110	53.0 %
11111	58.1 %

Table 2: Table showing that some patterns are highly predictive of price movements.

Doing so, and then normalising in a similiar way along columns any remaining statistics such as variance, mean, total volume etc we can then apply machine learning

However in practice, when doing so, we actually find our performance in terms of validation accuracy on the data was better when we simply normalise along columns, using scipy's scaling package. Normalising along rows we only seems to be able to achieve accuracy of around 71%, and so we actually abandon the method described above.

1.2.7 New Features Conclusion

Unfortunately, although we learn a lot about the data from our feature engineering, we are unable to tease out additional performance in terms of validation accuracy when adding these statistics to our data as additional columns. We hypothesise that this may be due to the prediction provided by the statistics overwhelming potentially more subtle information included in the data that we have not picked up on, meaning we don't receive significant performance improvements beyond the accuracy we identified already within those features.

We therefore don't include any of the features in the model we train on below, except for the level 1 volume split.

However we do take some comfort, in that when we made a simple deep learning model of a two layers with Relu and dropout, on just the three main features we identified (Pattern chance, Split between the level 1 bid and ask, and the stock label) that we achieved a 70.3 % validation acccuracy, and so have in effect achieved significant feature reduction.

1.3 Model Training

Having identified various potential features which may improve the performance of our model, and including the 'level 1 volume split', we now compile and train our feedforward neural network. We use the `[keras]` [1] package to do so. First, we randomly split `Data_A` the training data using the `sklearn.model_selection` package, opting for an 80:20 to give a reasonably large validation set. The accuracy of the model, defined by

$$\text{Accuracy} = \frac{\text{Number of correctly predicted labels}}{\text{Total number of labels}}, \tag{4}$$

gives us an indication of its performance.

It is common practice when training neural networks to standardise data to improve performance. The data exhibits significant variations in each column, this is not suprising as we are looking at financial returns data of stocks. Given this, features with different scales can dominate in the learning process. This is counter productive when trying to construct a robust model, and so we circumvent this issue by scaling each column using the `sklearn.preproccessing`[4] package, which takes the mean and standard deviation of feature X as

$$\frac{X-\mu}{\sigma}. \tag{5}$$

More practically, the typical activation functions (relu, sigmoid, tanh etc) used in neural networks are most effective when the inputs are zero-centered wihtin a reasonably small range.

Another practical regularisation technique used to improve neural network performance is dropout. During training, droupout randomly sets a proportion of the neurons to zero, with the purpose of preventing the model from to heavily relying on those which have been deactivated. This ensures that the model remains generalisable, and should ensure better performance on the test set.

We optimise the hyperparameters of our model using the `KerasTuner` [3] package, where we specifically optimise the number of layers, neutrons per layer, activation functions, drop out rate and learning rate of our model. Using `KerasTuner`, we are able to set predefined values for the five different model hyper parameters we wish to optimise. `KerasTuner` then randomly iterates through different combinations of the values, training the model for each combination and tracking the combination with the best performing accuracy on the dataset. To keep the search computationally feasible, we test ten random combinations of hyper parameters, training the model twice per combination and choose twenty epochs, meaning the model sees the dataset twenty times during training.

Throughout the training, the Adaptive Moment Estimation (Adam) optimisation algorithm is used. It is a gradient descent algorithm that is commonly used in deep learning, incorporating momentum and adaptive learning rates to efficiently converge to an optimal solution.

The optimised parameters are displayed in Table 3, and visualsied in Figure 4.

Hyper parameter	**Values Tested**	**Optimized Value**
Number of hidden layers	1, 2, 3	3
Neurons per hidden layer	32-1024	736 (Lay. 1), 576 (Lay. 2), 512 (Lay. 3)
Activation function	ReLU, tanh	ReLU
Dropout rate	0.1, 0.2, 0.3, 0.4, 0.5	0.5
Learning rate	0.01, 0.001, 0.0001	1×10^{-2}

Table 3: Hyper parameter Optimisation Results

We find that 3 layers is optimal for the model. Intuitively, this seems reasonable, having fewer layers risks under-fitting the data, whilst, as displayed by our feature engineering in the previous section, the relationship between `Data_A's` features and label appears relatively simple, and so having more than three layers risks over-fitting. Our model then has a final output layer, which uses a sigmoid activation function,

$$g(x) = \frac{1}{1 + e^{-x}}. \tag{6}$$

The sigmoid function is commonly used in binary classification [2] as it maps any input into a value between zero and one, with $\lim_{x\to\infty} g(x) = 1$ and $\lim_{x\to-\infty} g(x) = 0$. Hence it can be interpreted as a probability, representing the model's confidence that a sample's label will equal one. If the sigmoid function's output is above 0.5 the model predicts the sample's label as one, or the model predicts the sample's label as zero if the sigmoid function's output is below 0.5.

Our optimised model 4, displayed in figure 4, has 736 neurons in the first layer, 576 neurons in the second layer and 512 neurons in the third layer. Many neural networks are designed such that the neurons per layer decrease with each subsequent layer. This aims to combine low level features established in early layers into fewer, high-level features in the end layers, and our model follows this same structure. 736, 536 and 512 neurons per layer is relatively large given the size of `Data_A` and could cause over-fitting, but this is mitigated by the inclusion of regularisation in our model through dropout. Our optimised dropout rate of 0.5, which is particularly large. This implies that the neruons become highly codependent during training without dropout, and so the large dropout is ensuring that the model is learning general features of the data.

The Rectified Linear Unit function $\mathrm{ReLU}(x) = \max(0, x)$ was chosen as the optimal activation function in the model. The function was made popular by Glorot et al.(2011)[2] and has largely displaced the sigmoid and hyperbolic tangent function in hidden layers due to the simplicity of the function and its derivative.

With these optimal hyper parameters we achieve an accuracy of around 72.1% on `Data_A's` validation set with with scaling only of the original data, an improvement on the 70.3% achieved through our feature engineering.

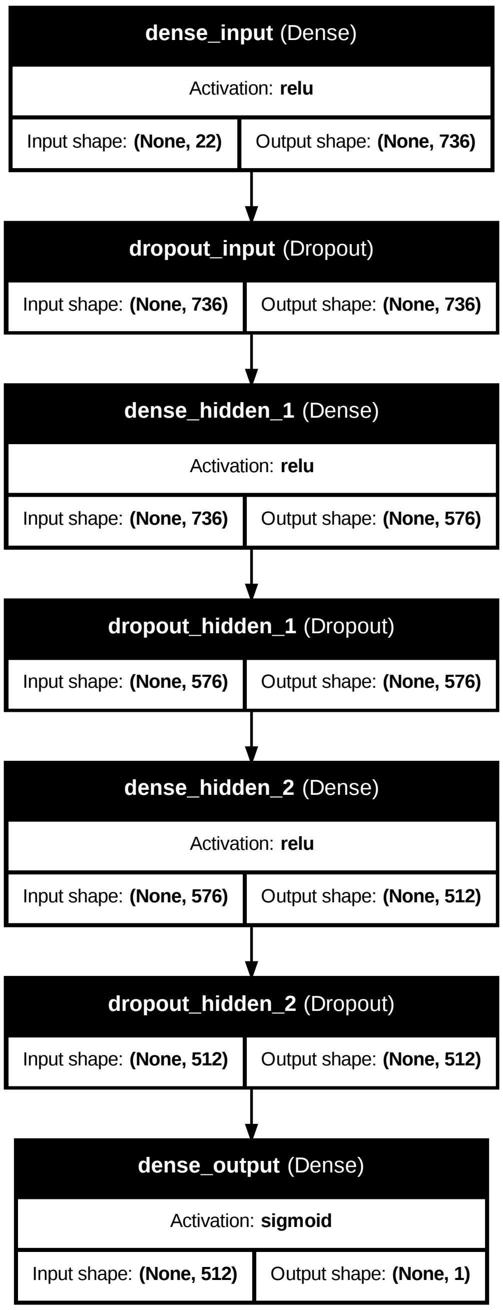

Figure 4: Caption

2 Conclusion

We have conducted analysis to understand the data set, and developed new features to add to the training. However when training our network, we achieved the best results when leaving these features out bar 'level 1 volume split'. When training we obtained optimal hyper parameters that achieve an accuracy of around 72% on `Data_A's` validation set.

This points to the ability of deep learning in learning patterns in data without formal identification or assistance prior to analysis. Specifically this highlights the ability for machine learning to pick out patterns that may usually have been missed or not well understood by a human observer.

3 Appendix

Below is our python code for both the feature engineering and model training

Modeltraining.py

```
!pip install --upgrade tensorflow
import numpy as np
import numpy.random as npr
import tensorflow.keras as keras
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense, Dropout, Input
from sklearn.utils.class_weight import compute_class_weight

plt.style.use('ggplot')



# extract data


column_names = ['midprice change direction', 'limit order level 1 ask price',
'limit order level 1 ask volume', 'limit order level 1 bid price',
'limit order level 1 bid volume', 'limit order level 2 ask price',
'limit order level 2 ask volume', 'limit order level 2 bid price',
'limit order level 2 bid volume', 'limit order level 3 ask price',
'limit order level 3 ask volume', 'limit order level 3 bid price',
'limit order level 3 bid volume', 'limit order level 4 ask price',
'limit order level 4 ask volume', 'limit order level 4 bid price',
'limit order level 4 bid volume', 'previous midprice change 1',
'previous midprice change 2', 'previous midprice change 3',
'previous midprice change 4', 'previous midprice change 5']
```

```
training_data =
pd.read_csv(r"C:\Users\charl\OneDrive\Documents\DeepLearning\Data_A.csv",
names=column_names)
test_data =
pd.read_csv(r"C:\Users\charl\OneDrive\Documents\DeepLearning\Data_B_nolabels.csv",
names=column_names)

from kerastuner.tuners import RandomSearch
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Input
from tensorflow.keras.optimizers import Adam
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Split the data
labels = training_data.iloc[:, 0].values
# Keep as a DataFrame for feature engineering if needed
features = training_data.iloc[:, 1:]

# add volume split
features['level 1 volume split'] = features['limit order level 1 bid volume'] -
features['limit order level 1 ask volume']


# split data

features_train, features_val, labels_train, labels_val =
train_test_split(features, labels, test_size=0.2, random_state=42)

# Scale the data
scaler = StandardScaler()
features_train = scaler.fit_transform(features_train)
features_val = scaler.transform(features_val)

# HyperModel: Function to build model with hyperparameters
def build_model(hp):
    model = Sequential()
    model.add(Input(shape=(features_train.shape[1],)))

    # Tune the number of layers and units in each layer
    for i in range(hp.Int('num_layers', 1, 3)):
        model.add(Dense(units=hp.Int(f'units_{i}', min_value=32, max_value=1024, step=32),
                        activation=hp.Choice('activation', ['relu', 'tanh'])))
        model.add(Dropout(rate=hp.Float('dropout', 0.1, 0.5, step=0.1)))

    # Output layer
    model.add(Dense(1, activation='sigmoid'))
    # Tune the learning rate for the optimizer
    learning_rate = hp.Choice('learning_rate', values=[1e-2, 1e-3, 1e-4])
    model.compile(optimizer=Adam(learning_rate=learning_rate),
                  loss='binary_crossentropy',
                  metrics=['accuracy'])

    return model

# Instantiate the tuner
tuner = RandomSearch(
    build_model,
    objective='val_accuracy',
    max_trials=10,  # Total number of different models to try
    executions_per_trial=2,  # Number of times to train each model to account for variance
    directory='tuning_results',
    project_name='midprice_prediction'
)

# Run the tuner
tuner.search(features_train, labels_train,
             validation_data=(features_val, labels_val),
             epochs=10,
             batch_size=64)

# Get the best hyperparameters
best_hps = tuner.get_best_hyperparameters(num_trials=1)[0]

# Print the best hyperparameters
print(f"""
The optimal number of layers is {best_hps.get('num_layers')},
the optimal number of units per layer is {best_hps.get('units_0')} to
{best_hps.get('units_2')} depending on layers,
the best activation function is {best_hps.get('activation')},
the best dropout rate is {best_hps.get('dropout')},
and the best learning rate is {best_hps.get('learning_rate')}.
""")

# Build and train the best model
best_model = tuner.hypermodel.build(best_hps)
history = best_model.fit(features_train, labels_train,
                         validation_data=(features_val, labels_val),
                         epochs=20,
                         batch_size=64)

# Evaluate the model
loss, accuracy = best_model.evaluate(features_val, labels_val)
print(f"Validation Loss: {loss:.4f}, Validation Accuracy: {accuracy:.4f}")
```

PredictingB.py

We then wish to predict the values in the unlabelled data based on our trained model. We have a number of points to check whether data has been processed as expected, and whether the final output is sensible based on our identified 'level 1 volume split' statistic. We found that it agreed with the predictions made by the algorithm 78 % of the time, giving us some comfort that the predictions are at least somewhat robust.

```
import tensorflow as tf
from tensorflow.keras.models import load_model
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

model = tf.keras.models.load_model('best_model.keras')

column_names = ['limit order level 1 ask price',
'limit order level 1 ask volume', 'limit order level 1 bid price',
'limit order level 1 bid volume', 'limit order level 2 ask price',
'limit order level 2 ask volume', 'limit order level 2 bid price',
'limit order level 2 bid volume', 'limit order level 3 ask price',
'limit order level 3 ask volume', 'limit order level 3 bid price',
'limit order level 3 bid volume', 'limit order level 4 ask price',
'limit order level 4 ask volume', 'limit order level 4 bid price',
'limit order level 4 bid volume', 'previous midprice change 1',
'previous midprice change 2', 'previous midprice change 3',
'previous midprice change 4', 'previous midprice change 5']

training_data = pd.read_csv("Data_A.csv",names=['midprice change direction'] +
column_names)
unlabelled_data = pd.read_csv("Data_B_nolabels.csv", names = column_names)

#Check data looks sensible
print("Initial Import of Data")
print("Training Data:")
print(training_data)
print("****************************")
print("Unlabelled Data")
print(unlabelled_data)
input("If data looks sensible, press enter:")
print("****************************")

# Split the data
labels = training_data.iloc[:, 0].values
features = training_data.iloc[:, 1:]

# Add test statistic that was deemed useful
features['level 1 volume split'] = features['limit order level 1 bid volume'] -
features['limit order level 1 ask volume']
unlabelled_data['level 1 volume split'] = unlabelled_data['limit order level 1
bid volume'] - unlabelled_data['limit order level 1 ask volume']

#Check data looks sensible
print("We have now added the level 1 volume split")
print("Training Data:")
print(features)
print("****************************")
print("Unlabelled Data")
print(unlabelled_data)
input("If data looks sensible, press enter:")
print("****************************")
# split data
features_train, features_val, labels_train, labels_val =
train_test_split(features, labels, test_size=0.2, random_state=42)

#Check data looks sensible
print("We have now split the training data")
print("Training Data:")
print(features_train)
print("****************************")
print("Unlabelled Data")
print(unlabelled_data)
input("If data looks sensible, press enter:")
print("****************************")
input("If data looks sensible, press enter:")
print("****************************")

# Scale the data
scaler = StandardScaler()
features_train_scaled = scaler.fit_transform(features_train)
features_val_scaled = scaler.transform(features_val)
unlabelled_data_scaled = scaler.transform(unlabelled_data)

#Check data looks sensible
print("Training Data:")
print(features_train_scaled)
print("****************************")
print("Unlabelled Data")
print(unlabelled_data_scaled)
input("If data looks sensible, press enter:")

# Check it's working on the training data
training_predictions = model.predict(features_train_scaled)

x = np.absolute(np.transpose(np.rint(training_predictions)) - labels_train)
print("Total Number Misidentified")
print(np.sum(x))
print("Percentage Incorrect")
print(np.sum(x)/160000)
print("Validation Accuracy")
print(1 - np.sum(x)/160000)

# Got 72%, which is what we expected

# # Code below is broken for some reason
# training_predictions = np.rint(training_predictions)
# training_predictions = pd.DataFrame(training_predictions, columns = ['midprice
change prediction'])
# print("Accuracy of 'level 1 volume split' stat")
# print(((training_predictions['midprice change prediction'] -
(features_train['level 1 volume split'] > 0)) == 0).sum()/160000)

# Check it's working on the validation data
val_predictions = model.predict(features_val_scaled)

x = np.absolute(np.transpose(np.rint(val_predictions)) - labels_val)
print("Total Number Misidentified")
print(np.sum(x))
print("Percentage Incorrect")
print(np.sum(x)/40000)
print("Validation Accuracy")
print(1 - np.sum(x)/40000)

# # Code below is broken for some reason
# val_predictions = np.rint(val_predictions)
# val_predictions = pd.DataFrame(val_predictions, columns = ['midprice change
prediction'])
# print("Accuracy of 'level 1 volume split' stat")
# print(((val_predictions['midprice change prediction'] - (features_val['level 1
volume split'] > 0)) == 0).sum()/40000)

# Got 72%, which is what we expected

# Apply to unlabelled data
val_predictions = model.predict(unlabelled_data_scaled)
predictions = np.rint(val_predictions)

predictions = pd.DataFrame(predictions, columns = ['midprice change prediction'])

print("Check predictions are sensible:")
print(predictions)
print(unlabelled_data)

# Check based on test statistic we know works well to see if labels are sensible
print("Accuracy of 'level 1 volume split' stat")
print(((predictions['midprice change prediction'] - (unlabelled_data['level 1
volume split'] > 0)).astype(int) == 0).sum()/20000)
```

```
input("If data looks sensible, press enter:")

# Outputs
# Outputs
predictions.to_csv('DataBPredictionsFinal.txt', sep='\n', index=False,
header=False)
```

Graphproduction.py

We recreate various graphs included in the report, including analysis of certain features we identified.

```
import pandas as pd
import matplotlib.pyplot as plt
from Data_Cleaning2 import add_distribution_statistics, book_volume_statistics

# Set up some sensible column names for our columns
column_names = ['limit order level 1 ask price',
'limit order level 1 ask volume','limit order level 1 bid price',
'limit order level 1 bid volume',
'limit order level 2 ask price', 'limit order level 2 ask volume',
'limit order level 2 bid price', 'limit order level 2 bid volume',
'limit order level 3 ask price', 'limit order level 3 ask volume',
'limit order level 3 bid price', 'limit order level 3 bid volume',
'limit order level 4 ask price', 'limit order level 4 ask volume',
'limit order level 4 bid price', 'limit order level 4 bid volume',
'previous midprice change 1', 'previous midprice change 2',
'previous midprice change 3', 'previous midprice change 4',
'previous midprice change 5']

# Import the data
unlabelled_raw_data = pd.read_csv("Data_B_nolabels.csv", names = column_names)
labelled_raw_data = pd.read_csv("Data_A.csv",
names = ['midprice change direction'] + column_names)

unlabelled_raw_data = add_distribution_statistics(unlabelled_raw_data)
unlabelled_raw_data = book_volume_statistics(unlabelled_raw_data)
# Marking the unlabelled data as 2 for comparison
unlabelled_raw_data['midprice change direction'] = 2
unlabelled_training_objective = unlabelled_raw_data['midprice change direction']

labelled_training_data = labelled_raw_data

labelled_training_data = add_distribution_statistics(labelled_training_data)
labelled_training_data = book_volume_statistics(labelled_training_data)
labelled_training_objective = labelled_training_data['midprice change direction']
```

```
# Bar chart of the volume in a distribution

prices = [labelled_raw_data.loc[0, 'limit order level 4 ask price'],
labelled_raw_data.loc[0, 'limit order level 3 ask price'],
labelled_raw_data.loc[0, 'limit order level 2 ask price'],
labelled_raw_data.loc[0, 'limit order level 1 ask price'],
labelled_raw_data.loc[0, 'limit order level 1 bid price'],
labelled_raw_data.loc[0, 'limit order level 2 bid price'],
labelled_raw_data.loc[0, 'limit order level 3 bid price'],
labelled_raw_data.loc[0, 'limit order level 4 bid price']]
volumes = [labelled_raw_data.loc[0, 'limit order level 4 ask volume'],
labelled_raw_data.loc[0, 'limit order level 3 ask volume'],
labelled_raw_data.loc[0, 'limit order level 2 ask volume'],
labelled_raw_data.loc[0, 'limit order level 1 ask volume'],
labelled_raw_data.loc[0, 'limit order level 1 bid volume'],
labelled_raw_data.loc[0, 'limit order level 2 bid volume'],
labelled_raw_data.loc[0, 'limit order level 3 bid volume'],
labelled_raw_data.loc[0, 'limit order level 4 bid volume']]

plt.figure(figsize = (10,6))
plt.bar(prices, volumes, width = 30)
plt.xticks(ticks = [692500, 692600, 692700, 692800, 692900, 693000,
693100, 693200, 693300, 693400, 693500])
plt.xlabel('Bid/Ask Prices')
plt.ylabel('Volume')
plt.title('Distribution of Bid Asks for the first Data Point')
plt.grid()
plt.savefig('Pricedistribution.png')
plt.show()


# Producing graph showing Data Point Classification
plt.figure(figsize=(10, 6))
plt.scatter(unlabelled_raw_data['overall mean'],
unlabelled_raw_data['midprice change direction'], alpha = 0.01,
label = 'Midprice Change Variable from Labelled Data')
plt.scatter(labelled_training_data['overall mean'],
labelled_training_data['midprice change direction'], alpha = 0.01,
label = 'Unlabelled Data')
# Custom x-axis labels
y_labels = ['Midprice Down', 'Midprice Up', 'Unlabelled Data']
# Change the x-tick positions and labels

plt.yticks(ticks=[0,1,2], labels=y_labels)
plt.ylim(-3, 5)
plt.xlabel('Mean Price')
plt.ylabel('Data Point Classification')
plt.title('Mean Price of Unlabelled Data, and Labelled Data by Midprice Movement
Label')
plt.grid()
plt.savefig('Meanfigure.png')
plt.show()


# Analysis of effectiveness of volume split statistic at predicting movements,
# split based on mean of data.

print("Volume Split on Whole Data Set")
print((((labelled_raw_data['level 1 volume split'] > 0).astype(int) -
labelled_training_objective) == 0).sum()/labelled_training_objective.count())
print((((labelled_raw_data['level 2 volume split'] > 0).astype(int) -
labelled_training_objective) == 0).sum()/labelled_training_objective.count())
print((((labelled_raw_data['level 3 volume split'] > 0).astype(int) -
labelled_training_objective) == 0).sum()/labelled_training_objective.count())
print((((labelled_raw_data['level 4 volume split'] > 0).astype(int) -
labelled_training_objective) == 0).sum()/labelled_training_objective.count())
print((((labelled_raw_data['overall volume split'] > 0).astype(int) -
labelled_training_objective) == 0).sum()/labelled_training_objective.count())

# Producing graph showing Data Point Classification
plt.figure(figsize=(10, 6))
plt.scatter(labelled_raw_data['level 1 skew'],
labelled_raw_data['midprice change direction'], alpha = 0.02,
label = 'Midprice Change Variable from Labelled Data')
# Custom x-axis labels
y_labels = ['Midprice Down', 'Midprice Up']
# Change the x-tick positions and labels
plt.yticks(ticks=[0,1], labels=y_labels)
plt.ylim(-3, 5)
plt.xlim(-40,40)
plt.xlabel('Level 1 Skew')
plt.title('Level 1 Skew of Labelled Data by Midprice Movement Label')
plt.grid()
plt.savefig('Skewfigure.png')
plt.show()

print("**************************************")

# Creating splits in data
high_data = labelled_raw_data[labelled_raw_data['overall mean'] > 900000]
middle_data = labelled_raw_data[labelled_raw_data['overall mean'] <= 900000]
upper_middle_data = middle_data[middle_data['overall mean'] > 795000]
middle_middle_data = labelled_raw_data[labelled_raw_data['overall mean'] <= 795000]
middle_middle_data = middle_middle_data[middle_middle_data['overall mean'] > 707500]
middle_data = labelled_raw_data[labelled_raw_data['overall mean'] > 550000]
lower_middle_data = middle_data[middle_data['overall mean'] <= 707500]
lower_data = labelled_raw_data[labelled_raw_data['overall mean'] <= 550000]
```

```
upper_lower_data = lower_data[lower_data['overall mean'] > 465000]
lower_lower_data = lower_data[lower_data['overall mean'] <= 465000]

print("High Data")
labelled_raw_data = high_data
labelled_training_objective = labelled_raw_data['midprice change direction']



print((((labelled_raw_data['level 1 volume split'] > 0).astype(int) -
labelled_training_objective) == 0).sum()/labelled_training_objective.count())
print((((labelled_raw_data['level 2 volume split'] > 0).astype(int) -
labelled_training_objective) == 0).sum()/labelled_training_objective.count())
print((((labelled_raw_data['level 3 volume split'] > 0).astype(int) -
labelled_training_objective) == 0).sum()/labelled_training_objective.count())
print((((labelled_raw_data['level 4 volume split'] > 0).astype(int) -
labelled_training_objective) == 0).sum()/labelled_training_objective.count())
print((((labelled_raw_data['overall volume split'] > 0).astype(int) -
labelled_training_objective) == 0).sum()/labelled_training_objective.count())
print("**************************************")
print("Middle Data")
labelled_raw_data = middle_data
labelled_training_objective = labelled_raw_data['midprice change direction']



print((((labelled_raw_data['level 1 volume split'] > 0).astype(int) -
labelled_training_objective) == 0).sum()/labelled_training_objective.count())
print((((labelled_raw_data['level 2 volume split'] > 0).astype(int) -
labelled_training_objective) == 0).sum()/labelled_training_objective.count())
print((((labelled_raw_data['level 3 volume split'] > 0).astype(int) -
labelled_training_objective) == 0).sum()/labelled_training_objective.count())
print((((labelled_raw_data['level 4 volume split'] > 0).astype(int) -
labelled_training_objective) == 0).sum()/labelled_training_objective.count())
print((((labelled_raw_data['overall volume split'] > 0).astype(int) -
labelled_training_objective) == 0).sum()/labelled_training_objective.count())
print("**************************************")

print("Upper Middle Data")
labelled_raw_data = upper_middle_data
labelled_training_objective = labelled_raw_data['midprice change direction']



print((((labelled_raw_data['level 1 volume split'] > 0).astype(int) -
labelled_training_objective) == 0).sum()/labelled_training_objective.count())
print((((labelled_raw_data['level 2 volume split'] > 0).astype(int) -
labelled_training_objective) == 0).sum()/labelled_training_objective.count())
print((((labelled_raw_data['level 3 volume split'] > 0).astype(int) -
labelled_training_objective) == 0).sum()/labelled_training_objective.count())
print((((labelled_raw_data['level 4 volume split'] > 0).astype(int) -
labelled_training_objective) == 0).sum()/labelled_training_objective.count())
print((((labelled_raw_data['overall volume split'] > 0).astype(int) -
labelled_training_objective) == 0).sum()/labelled_training_objective.count())
print("**************************************")
print("Middle Middle Data")
labelled_raw_data = middle_middle_data
labelled_training_objective = labelled_raw_data['midprice change direction']


print((((labelled_raw_data['level 1 volume split'] > 0).astype(int) -
labelled_training_objective) == 0).sum()/labelled_training_objective.count())
print((((labelled_raw_data['level 2 volume split'] > 0).astype(int) -
labelled_training_objective) == 0).sum()/labelled_training_objective.count())
print((((labelled_raw_data['level 3 volume split'] > 0).astype(int) -
labelled_training_objective) == 0).sum()/labelled_training_objective.count())
print((((labelled_raw_data['level 4 volume split'] > 0).astype(int) -
labelled_training_objective) == 0).sum()/labelled_training_objective.count())
print((((labelled_raw_data['overall volume split'] > 0).astype(int) -
labelled_training_objective) == 0).sum()/labelled_training_objective.count())
print("**************************************")

print("Lower Middle Data")
labelled_raw_data = lower_middle_data
labelled_training_objective = labelled_raw_data['midprice change direction']


print((((labelled_raw_data['level 1 volume split'] > 0).astype(int) -
labelled_training_objective) == 0).sum()/labelled_training_objective.count())
print((((labelled_raw_data['level 2 volume split'] > 0).astype(int) -
labelled_training_objective) == 0).sum()/labelled_training_objective.count())
print((((labelled_raw_data['level 3 volume split'] > 0).astype(int) -
labelled_training_objective) == 0).sum()/labelled_training_objective.count())
print((((labelled_raw_data['level 4 volume split'] > 0).astype(int) -
labelled_training_objective) == 0).sum()/labelled_training_objective.count())
print((((labelled_raw_data['overall volume split'] > 0).astype(int) -
labelled_training_objective) == 0).sum()/labelled_training_objective.count())
print("**************************************")

print("Lower Data")
labelled_raw_data = lower_data
labelled_training_objective = labelled_raw_data['midprice change direction']


print((((labelled_raw_data['level 1 volume split'] > 0).astype(int) -
labelled_training_objective) == 0).sum()/labelled_training_objective.count())
print((((labelled_raw_data['level 2 volume split'] > 0).astype(int) -
labelled_training_objective) == 0).sum()/labelled_training_objective.count())
```

```
print((((labelled_raw_data['level 3 volume split'] > 0).astype(int) -
labelled_training_objective) == 0).sum()/labelled_training_objective.count())
print((((labelled_raw_data['level 4 volume split'] > 0).astype(int) -
labelled_training_objective) == 0).sum()/labelled_training_objective.count())
print((((labelled_raw_data['overall volume split'] > 0).astype(int) -
labelled_training_objective) == 0).sum()/labelled_training_objective.count())
print("***************************************")

print("Upper Lower Data")
labelled_raw_data = upper_lower_data
labelled_training_objective = labelled_raw_data['midprice change direction']


print((((labelled_raw_data['level 1 volume split'] > 0).astype(int) -
labelled_training_objective) == 0).sum()/labelled_training_objective.count())
print((((labelled_raw_data['level 2 volume split'] > 0).astype(int) -
labelled_training_objective) == 0).sum()/labelled_training_objective.count())
print((((labelled_raw_data['level 3 volume split'] > 0).astype(int) -
labelled_training_objective) == 0).sum()/labelled_training_objective.count())
print((((labelled_raw_data['level 4 volume split'] > 0).astype(int) -
labelled_training_objective) == 0).sum()/labelled_training_objective.count())
print((((labelled_raw_data['overall volume split'] > 0).astype(int) -
labelled_training_objective) == 0).sum()/labelled_training_objective.count())
print("***************************************")

print("Lower Lower Data")
labelled_raw_data = lower_lower_data
labelled_training_objective = labelled_raw_data['midprice change direction']


print((((labelled_raw_data['level 1 volume split'] > 0).astype(int) -
labelled_training_objective) == 0).sum()/labelled_training_objective.count())
print((((labelled_raw_data['level 2 volume split'] > 0).astype(int) -
labelled_training_objective) == 0).sum()/labelled_training_objective.count())
print((((labelled_raw_data['level 3 volume split'] > 0).astype(int) -
labelled_training_objective) == 0).sum()/labelled_training_objective.count())
print((((labelled_raw_data['level 4 volume split'] > 0).astype(int) -
labelled_training_objective) == 0).sum()/labelled_training_objective.count())
print((((labelled_raw_data['overall volume split'] > 0).astype(int) -
labelled_training_objective) == 0).sum()/labelled_training_objective.count())
print("***************************************")

# This tells us pretty clearly that they operate in two modes
# probably the first stock has a mean and prices above 550k,
# and the second below 550k
# It might be worth training the algorithm separately on each.
```

Changepattercalc.py

We compute the midprice change patterns total frequency, as seen in Table 2.

```
import pandas as pd

column_names = ['midprice change direction', 'limit order level 1 ask price'
, 'limit order level 1 ask volume', 'limit order level 1 bid price',
'limit order level 1 bid volume','limit order level 2 ask price',
'limit order level 2 ask volume','limit order level 2 bid price',
'limit order level 2 bid volume', 'limit order level 3 ask price',
'limit order level 3 ask volume','limit order level 3 bid price',
'limit order level 3 bid volume','limit order level 4 ask price',
'limit order level 4 ask volume','limit order level 4 bid price',
'limit order level 4 bid volume', 'previous midprice change 1',
'previous midprice change 2','previous midprice change 3',
'previous midprice change 4', 'previous midprice change 5']

raw_data = pd.read_csv("Data_A.csv", names = column_names)

# We create strings of price changes
raw_data['price change string'] =
raw_data['previous midprice change 1'].astype(str) +
raw_data['previous midprice change 2'].astype(str) +
raw_data['previous midprice change 3'].astype(str) +
raw_data['previous midprice change 4'].astype(str) +
raw_data['previous midprice change 5'].astype(str)

# Then group price changes by the mean movement, to calculate frequencies.
pattern_chance = raw_data.groupby(
'price change string').mean()['midprice change direction']

# Save to a CSV so we don't have to compute it every time
pattern_chance.to_csv('changepatternchance.csv', index=True)

# Display for analysis
print(pattern_chance)


# Checking if this feature is robust. If we reverse the values, we should
# approximately get 1 - the frequency. So adding together should be approximately
# 1 in each entry.
copy = pattern_chance.copy()

for i in range(32):
    copy[i] = pattern_chance[i] + pattern_chance[31-i]

print(copy)

# And we see that is true.
```

Data_Cleaning2.py

We define various functions to create new features, as well as some normalisation based on the distribution we identified, although these methods are not used in the final algorithm except to understand the data better.

```
#Cleaning and processing the data to see if we can gain insights

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# A starting guess is that just mean reversion (based on stylistic facts) might
# occur. Therefore we could try this as a method of prediction.
# If the sum is 5, we would guess the next would be 0. 4, 0 but a bit less, 0 and
# we would guess 1 pretty strongly. So we'll use a formula of (5 - sum)/5. If the
# sum is 5 we are saying very non rigorously, that there is a chance of 1 of the
# asset going up

# Function to add mean reversion to training data
def mean_reversion(training_data):

    training_data['mean reversion'] = (5 -
    training_data[['previous midprice change 1',
    'previous midprice change 2', 'previous midprice change 3',
    'previous midprice change 4', 'previous midprice change 5']].sum(axis = 1))/5

    return training_data

## When testing mean reversion as below, we obtain slightly better than chance
# print(training_data.head())
# print(training_objective.head())
# print((((training_data['mean reversion'] > 0.5) - training_objective) ==
0).sum()/180000)

# Function to add midprice change direction as previously computed to our data
def midprice_change_direction(training_data):

    pattern_chance = pd.read_csv('changepatternchance.csv')
    pattern_chance = pattern_chance.set_index('price change string')



    training_data['price change string'] =
    training_data['previous midprice change 1'].astype(str) +
    training_data['previous midprice change 2'].astype(str) +
    training_data['previous midprice change 3'].astype(str) +
    training_data['previous midprice change 4'].astype(str) +
    training_data['previous midprice change 5'].astype(str)
    training_data['price change string'] = training_data['price change string'].astype(int)
    # So we add to our training data our useful predicted chance based on this,
    # and drop the strings, which are non-numeric (since the chances for each of
    # the strings is unique, this also serves as an encoding of the pattern)

    training_data['price pattern chance'] = 0
    for index in training_data.index:
        training_data.at[index, 'price pattern chance'] =
        pattern_chance['midprice change direction'].loc[training_data['price change string']
    # print(training_data.head())

    training_data = training_data.drop('price change string', axis = 1)

    #print(training_data.head())

    return training_data


# Compute and add various statistics that we identified as potentially informative
# to the training data
def add_distribution_statistics(training_data):

    # Computing totals
    training_data['level 1 total'] = training_data['limit order level 1 bid price']*
    training_data['limit order level 1 bid volume'] +
    training_data['limit order level 1 ask price']*
    training_data['limit order level 1 ask volume']

    training_data['level 2 total'] = training_data['limit order level 2 bid price']*
    training_data['limit order level 2 bid volume'] +
    training_data['limit order level 2 ask price']*
    training_data['limit order level 2 ask volume']

    training_data['level 3 total'] = training_data['limit order level 3 bid price']*
    training_data['limit order level 3 bid volume'] +
    training_data['limit order level 3 ask price']*
    training_data['limit order level 3 ask volume']
    training_data['level 4 total'] = training_data['limit order level 4 bid price']*
    training_data['limit order level 4 bid volume'] +
    training_data['limit order level 4 ask price']*
    training_data['limit order level 4 ask volume']

    training_data['overall total'] = training_data['level 1 total'] +
    training_data['level 2 total'] + training_data['level 3 total'] +
    training_data['level 4 total']

    # Computing volumes
    training_data['level 1 volume'] = (training_data['limit order level 1 bid
    volume'] + training_data['limit order level 1 ask volume'])
```

```
training_data['level 2 volume'] = (training_data['limit order level 2 bid
volume'] + training_data['limit order level 2 ask volume'])
training_data['level 3 volume'] = (training_data['limit order level 3 bid
volume'] + training_data['limit order level 3 ask volume'])
training_data['level 4 volume'] = (training_data['limit order level 4 bid
volume'] + training_data['limit order level 4 ask volume'])
training_data['overall volume'] = training_data['level 1 volume'] +
training_data['level 2 volume'] + training_data['level 3 volume'] +
training_data['level 4 volume']

# Computing means
training_data['level 1 mean'] = training_data['level 1 total'] /
training_data['level 1 volume']
training_data['level 2 mean'] = training_data['level 2 total'] /
training_data['level 2 volume']
training_data['level 3 mean'] = training_data['level 3 total'] /
training_data['level 3 volume']
training_data['level 4 mean'] = training_data['level 4 total'] /
training_data['level 4 volume']
training_data['overall mean'] = training_data['overall total'] /
training_data['overall volume']

# Identifying stock
# Adding stock label
training_data['stock_label'] = (training_data['overall mean'] >
550000).astype(int)

# Computing Variances
training_data['level 1 variance'] = ((training_data['limit order level 1 bid
price'] - training_data['level 1 mean'])**2 * training_data['limit order level
1 bid volume'] + (training_data['limit order level 1 ask price'] -
training_data['level 1 mean'])**2 * training_data['limit order level 1 ask
volume'] )/training_data['level 1 volume']
training_data['level 2 variance'] = ((training_data['limit order level 2 bid
price'] - training_data['level 2 mean'])**2 * training_data['limit order level
2 bid volume'] + (training_data['limit order level 2 ask price'] -
training_data['level 2 mean'])**2 * training_data['limit order level 2 ask
volume'] )/training_data['level 2 volume']
training_data['level 3 variance'] = ((training_data['limit order level 3 bid
price'] - training_data['level 3 mean'])**2 * training_data['limit order level
3 bid volume'] + (training_data['limit order level 3 ask price'] -
training_data['level 3 mean'])**2 * training_data['limit order level 3 ask
volume'] )/training_data['level 3 volume']
training_data['level 4 variance'] = ((training_data['limit order level 4 bid
price'] - training_data['level 4 mean'])**2 * training_data['limit order level
4 bid volume'] + (training_data['limit order level 4 ask price'] -
training_data['level 4 mean'])**2 * training_data['limit order level 4 ask
volume'] )/training_data['level 4 volume']
training_data['overall variance'] = (training_data['level 1
variance']*training_data['level 1 volume'] + training_data['level 2
variance']*training_data['level 2 volume'] + training_data['level 3
variance']*training_data['level 3 volume'] + training_data['level 4
variance']*training_data['level 4 volume'])/training_data['overall volume']

# Computing Skews
training_data['level 1 skew'] = ((training_data['limit order level 1 bid
price'] - training_data['level 1 mean'])**3 * training_data['limit order level
1 bid volume'] + (training_data['limit order level 1 ask price'] -
training_data['level 1 mean'])**3 * training_data['limit order level 1 ask
volume'] )/training_data['level 1 volume']
training_data['level 2 skew'] = ((training_data['limit order level 2 bid
price'] - training_data['level 2 mean'])**3 * training_data['limit order level
2 bid volume'] + (training_data['limit order level 2 ask price'] -
training_data['level 2 mean'])**3 * training_data['limit order level 2 ask
volume'] )/training_data['level 2 volume']
training_data['level 3 skew'] = ((training_data['limit order level 3 bid
price'] - training_data['level 3 mean'])**3 * training_data['limit order level
3 bid volume'] + (training_data['limit order level 3 ask price'] -
training_data['level 3 mean'])**3 * training_data['limit order level 3 ask
volume'] )/training_data['level 3 volume']
training_data['level 4 skew'] = ((training_data['limit order level 4 bid
price'] - training_data['level 4 mean'])**3 * training_data['limit order level
4 bid volume'] + (training_data['limit order level 4 ask price'] -
training_data['level 4 mean'])**3 * training_data['limit order level 4 ask
volume'] )/training_data['level 4 volume']
training_data['overall skew'] = (training_data['level 1
skew']*training_data['level 1 volume'] + training_data['level 2
skew']*training_data['level 2 volume'] + training_data['level 3
skew']*training_data['level 3 volume'] + training_data['level 4
skew']*training_data['level 4 volume'])/training_data['overall volume']

training_data['level 1 skew'] = training_data['level 1 skew'] /
(np.sqrt(training_data['level 1 variance'])**3)
training_data['level 2 skew'] = training_data['level 2 skew'] /
(np.sqrt(training_data['level 2 variance'])**3)
training_data['level 3 skew'] = training_data['level 3 skew'] /
(np.sqrt(training_data['level 3 variance'])**3)
training_data['level 4 skew'] = training_data['level 4 skew'] /
(np.sqrt(training_data['level 4 variance'])**3)
training_data['overall skew'] = training_data['overall skew'] /
(np.sqrt(training_data['overall variance'])**3)

# Computing Kurtosis, as it might be useful
training_data['level 1 kurtosis'] = ((training_data['limit order level 1 bid
price'] - training_data['level 1 mean'])**4 * training_data['limit order level
1 bid volume'] + (training_data['limit order level 1 ask price'] -
training_data['level 1 mean'])**4 * training_data['limit order level 1 ask
volume'] )/training_data['level 1 volume']
```

```
    training_data['level 2 kurtosis'] = ((training_data['limit order level 2 bid
    price'] - training_data['level 2 mean'])**4 * training_data['limit order level
    2 bid volume'] + (training_data['limit order level 2 ask price'] -
    training_data['level 2 mean'])**4 * training_data['limit order level 2 ask
    volume'] )/training_data['level 2 volume']
    training_data['level 3 kurtosis'] = ((training_data['limit order level 3 bid
    price'] - training_data['level 3 mean'])**4 * training_data['limit order level
    3 bid volume'] + (training_data['limit order level 3 ask price'] -
    training_data['level 3 mean'])**4 * training_data['limit order level 3 ask
    volume'] )/training_data['level 3 volume']
    training_data['level 4 kurtosis'] = ((training_data['limit order level 4 bid
    price'] - training_data['level 4 mean'])**4 * training_data['limit order level
    4 bid volume'] + (training_data['limit order level 4 ask price'] -
    training_data['level 4 mean'])**4 * training_data['limit order level 4 ask
    volume'] )/training_data['level 4 volume']
    training_data['overall kurtosis'] = (training_data['level 1
    kurtosis']*training_data['level 1 volume'] + training_data['level 2
    kurtosis']*training_data['level 2 volume'] + training_data['level 3
    kurtosis']*training_data['level 3 volume'] + training_data['level 4
    kurtosis']*training_data['level 4 volume'])/training_data['overall volume']

    training_data['level 1 kurtosis'] = training_data['level 1 kurtosis'] /
    (np.sqrt(training_data['level 1 variance'])**4)
    training_data['level 2 kurtosis'] = training_data['level 2 kurtosis'] /
    (np.sqrt(training_data['level 2 variance'])**4)
    training_data['level 3 kurtosis'] = training_data['level 3 kurtosis'] /
    (np.sqrt(training_data['level 3 variance'])**4)
    training_data['level 4 kurtosis'] = training_data['level 4 kurtosis'] /
    (np.sqrt(training_data['level 4 variance'])**4)
    training_data['overall kurtosis'] = training_data['overall kurtosis'] /
    (np.sqrt(training_data['overall variance'])**4)

    return training_data

# We can also use the statistics we computed to normalise the data
def normalise_data(training_data):
    # Normalise prices, on some distribution
    training_data['limit order level 1 ask price'] = (training_data['limit order
    level 1 ask price'] - training_data['overall mean'])/training_data['overall
    variance']
    training_data['limit order level 2 ask price'] = (training_data['limit order
    level 2 ask price'] - training_data['overall mean'])/training_data['overall
    variance']
    training_data['limit order level 3 ask price'] = (training_data['limit order
    level 3 ask price'] - training_data['overall mean'])/training_data['overall
    variance']
    training_data['limit order level 4 ask price'] = (training_data['limit order
    level 4 ask price'] - training_data['overall mean'])/training_data['overall
    variance']
```

```
training_data['limit order level 1 bid price'] = (training_data['limit order
level 1 bid price'] - training_data['overall mean'])/training_data['overall
variance']
training_data['limit order level 2 bid price'] = (training_data['limit order
level 2 bid price'] - training_data['overall mean'])/training_data['overall
variance']
training_data['limit order level 3 bid price'] = (training_data['limit order
level 3 bid price'] - training_data['overall mean'])/training_data['overall
variance']
training_data['limit order level 4 bid price'] = (training_data['limit order
level 4 bid price'] - training_data['overall mean'])/training_data['overall
variance']



# We need to normalise volumes. We will just do over the total volume, so
# they represent proportions in some sense

training_data['limit order level 1 ask volume'] = training_data['limit order
level 1 ask volume'] / training_data['overall volume']
training_data['limit order level 2 ask volume'] = training_data['limit order
level 2 ask volume'] / training_data['overall volume']
training_data['limit order level 3 ask volume'] = training_data['limit order
level 3 ask volume'] / training_data['overall volume']
training_data['limit order level 4 ask volume'] = training_data['limit order
level 4 ask volume'] / training_data['overall volume']

training_data['limit order level 1 bid volume'] = training_data['limit order
level 1 bid volume'] / training_data['overall volume']
training_data['limit order level 2 bid volume'] = training_data['limit order
level 2 bid volume'] / training_data['overall volume']
training_data['limit order level 3 bid volume'] = training_data['limit order
level 3 bid volume'] / training_data['overall volume']
training_data['limit order level 4 bid volume'] = training_data['limit order
level 4 bid volume'] / training_data['overall volume']

training_data['level 1 volume split'] = training_data['level 1 volume split']
/ training_data['level 1 volume']
training_data['level 2 volume split'] = training_data['level 2 volume split']
/ training_data['level 2 volume']
training_data['level 3 volume split'] = training_data['level 3 volume split']
/ training_data['level 3 volume']
training_data['level 4 volume split'] = training_data['level 4 volume split']
/ training_data['level 4 volume']
training_data['overall volume split'] = training_data['overall volume split']

/ training_data['overall volume']
```

```
# Now we’ll just normalise everything that is pretty big
# Totals
training_data[’level 1 total’] = training_data[’level 1
total’]/training_data[’level 1 total’].abs().max()
training_data[’level 2 total’] = training_data[’level 2
total’]/training_data[’level 2 total’].abs().max()
training_data[’level 3 total’] = training_data[’level 3
total’]/training_data[’level 3 total’].abs().max()
training_data[’level 4 total’] = training_data[’level 4
total’]/training_data[’level 4 total’].abs().max()
training_data[’overall total’] = training_data[’overall
total’]/training_data[’overall total’].abs().max()

# Volumes
training_data[’level 1 volume’] = training_data[’level 1
volume’]/training_data[’level 1 volume’].abs().max()
training_data[’level 2 volume’] = training_data[’level 2
volume’]/training_data[’level 2 volume’].abs().max()
training_data[’level 3 volume’] = training_data[’level 3
volume’]/training_data[’level 3 volume’].abs().max()
training_data[’level 4 volume’] = training_data[’level 4
volume’]/training_data[’level 4 volume’].abs().max()
training_data[’overall volume’] = training_data[’overall
volume’]/training_data[’overall volume’].abs().max()

# Means
training_data[’level 1 mean’] = training_data[’level 1
mean’]/training_data[’level 1 mean’].abs().max()
training_data[’level 2 mean’] = training_data[’level 2
mean’]/training_data[’level 2 mean’].abs().max()
training_data[’level 3 mean’] = training_data[’level 3
mean’]/training_data[’level 3 mean’].abs().max()
training_data[’level 4 mean’] = training_data[’level 4
mean’]/training_data[’level 4 mean’].abs().max()
training_data[’overall mean’] = training_data[’overall
mean’]/training_data[’overall mean’].abs().max()

# Variance
training_data[’level 1 variance’] = training_data[’level 1
variance’]/training_data[’level 1 variance’].abs().max()
training_data[’level 2 variance’] = training_data[’level 2
variance’]/training_data[’level 2 variance’].abs().max()
training_data[’level 3 variance’] = training_data[’level 3
variance’]/training_data[’level 3 variance’].abs().max()
training_data[’level 4 variance’] = training_data[’level 4
variance’]/training_data[’level 4 variance’].abs().max()
training_data[’overall variance’] = training_data[’overall
variance’]/training_data[’overall variance’].abs().max()
```

```
    # Skew and Kurtosis are also quite large, even though they are potentially
    # already normalised, so let's scale them down to between 0 and 1
    training_data['level 1 skew'] = training_data['level 1
    skew']/training_data['level 1 skew'].abs().max()
    training_data['level 2 skew'] = training_data['level 2
    skew']/training_data['level 2 skew'].abs().max()
    training_data['level 3 skew'] = training_data['level 3
    skew']/training_data['level 3 skew'].abs().max()
    training_data['level 4 skew'] = training_data['level 4
    skew']/training_data['level 4 skew'].abs().max()
    training_data['overall skew'] = training_data['overall
    skew']/training_data['overall skew'].abs().max()

    training_data['level 1 kurtosis'] = training_data['level 1
    kurtosis']/training_data['level 1 kurtosis'].abs().max()
    training_data['level 2 kurtosis'] = training_data['level 2
    kurtosis']/training_data['level 2 kurtosis'].abs().max()
    training_data['level 3 kurtosis'] = training_data['level 3
    kurtosis']/training_data['level 3 kurtosis'].abs().max()
    training_data['level 4 kurtosis'] = training_data['level 4
    kurtosis']/training_data['level 4 kurtosis'].abs().max()
    training_data['overall kurtosis'] = training_data['overall
    kurtosis']/training_data['overall kurtosis'].abs().max()

    return training_data

# Function that just computes the simpler volume split statistics, based on our
# analysis of level 1 skew
def book_volume_statistics(training_data):
    training_data['level 1 volume split'] = training_data['limit order level 1 bid
    volume'] - training_data['limit order level 1 ask volume']
    training_data['level 2 volume split'] = training_data['limit order level 2 bid
    volume'] - training_data['limit order level 2 ask volume']
    training_data['level 3 volume split'] = training_data['limit order level 3 bid
    volume'] - training_data['limit order level 3 ask volume']
    training_data['level 4 volume split'] = training_data['limit order level 4 bid
    volume'] - training_data['limit order level 4 ask volume']
    training_data['overall volume split'] = training_data['level 1 volume split']
    + training_data['level 2 volume split'] + training_data['level 3 volume
    split'] + training_data['level 4 volume split']

    return training_data

# Function to transform the data
def transform_data(training_data):
    training_data = mean_reversion(training_data)
    training_data = alternating_sum(training_data)
    training_data = midprice_change_direction(training_data)
    training_data = add_distribution_statistics(training_data)
    training_data = book_volume_statistics(training_data)
    training_data = normalise_data(training_data)

    return training_data
```

featureengineeringdataprep.py

The code below is used to prepare the csv data used to train the minimal model where we use only three features and obtain 70.3% validation accuracy. In the full model, we don't add any of these features.

```
    import numpy as np
import matplotlib.pyplot as plt
import Data_Cleaning2
import pandas as pd
from sklearn.model_selection import train_test_split

# Set up some sensible column names for our columns
column_names = ['midprice change direction', 'limit order level 1 ask price',
'limit order level 1 ask volume', 'limit order level 1 bid price',
                'limit order level 1 bid volume', 'limit order level 2 ask price',
                'limit order level 2 ask volume', 'limit order level 2 bid price',
                'limit order level 2 bid volume', 'limit order level 3 ask price',
                'limit order level 3 ask volume', 'limit order level 3 bid price',
                'limit order level 3 bid volume', 'limit order level 4 ask price',
                'limit order level 4 ask volume', 'limit order level 4 bid price',
                'limit order level 4 bid volume', 'previous midprice change 1',
                'previous midprice change 2', 'previous midprice change 3',
                'previous midprice change 4', 'previous midprice change 5']

# Import the data
raw_data = pd.read_csv("Data_A.csv", names = column_names)
discriminator = Data_Cleaning2.add_distribution_statistics(raw_data)

# Import the data again to make sure it's not been altered by the above
raw_data = pd.read_csv("Data_A.csv", names = column_names)

stock_1 = raw_data[discriminator['overall mean'] > 550000]
stock_2 = raw_data[discriminator['overall mean'] <= 550000]

stock_1_labels = stock_1['midprice change direction']
stock_2_labels = stock_2['midprice change direction']

stock_1.drop(labels = ['midprice change direction'], axis = 1)
stock_1.drop(labels = ['midprice change direction'], axis = 1)
```

```
raw_data_labels = raw_data['midprice change direction']
raw_data = raw_data.drop(labels = ['midprice change direction'], axis = 1)

print(raw_data['stock_label'])
input("Press enter:")

seed = 42
# Split the data into training and validation sets
stock_1_training_data, stock_1_test_data, stock_1_training_objective,
stock_1_test_objective = train_test_split(stock_1
    , stock_1_labels, test_size=0.2, random_state=seed)

stock_2_training_data, stock_2_test_data, stock_2_training_objective,
stock_2_test_objective = train_test_split(stock_2
    , stock_2_labels, test_size=0.2, random_state=seed)

all_training_data, all_test_data, all_training_objective, all_test_objective =
train_test_split(raw_data
    , raw_data_labels, test_size=0.2, random_state=seed)



# # Split off the objectives
# stock_1_training_objective = stock_1_training_data['midprice change direction']
# stock_1_test_objective = stock_1_test_data['midprice change direction']
stock_1_training_data = stock_1_training_data.drop(labels = ['midprice change
direction'], axis = 1)
stock_1_test_data = stock_1_test_data.drop(labels = ['midprice change direction'],
axis = 1)

# stock_2_training_objective = stock_2_training_data['midprice change direction']
# stock_2_test_objective = stock_2_test_data['midprice change direction']
stock_2_training_data = stock_2_training_data.drop(labels = ['midprice change
direction'], axis = 1)
stock_2_test_data = stock_2_test_data.drop(labels = ['midprice change direction'],
axis = 1)


# Add statistics and normalise according to data cleaning
stock_1_training_data = Data_Cleaning2.transform_data(stock_1_training_data, drop
= True)
stock_1_test_data = Data_Cleaning2.transform_data(stock_1_test_data, drop = True)

stock_2_training_data = Data_Cleaning2.transform_data(stock_2_training_data, drop
= True)
stock_2_test_data = Data_Cleaning2.transform_data(stock_2_test_data, drop = True)

all_training_data = Data_Cleaning2.transform_data(all_training_data, drop = True)
all_test_data = Data_Cleaning2.transform_data(all_test_data, drop = True)
```

```
# Print out and send to csv

stock_1_training_data.to_csv('stock_1_training.csv', index = False)
stock_1_test_data.to_csv('stock_1_test.csv', index = False)
stock_1_training_objective.to_csv('stock_1_trainingobjective.csv', index = False)
stock_1_test_objective.to_csv('stock_1_testobjective.csv', index = False)

print(stock_1_training_data)
print(stock_1_test_data)
print(stock_1_training_objective)
print(stock_1_test_objective)

stock_2_training_data.to_csv('stock_2_training.csv', index = False)
stock_2_test_data.to_csv('stock_2_test.csv', index = False)
stock_2_training_objective.to_csv('stock_2_trainingobjective.csv', index = False)
stock_2_test_objective.to_csv('stock_2_testobjective.csv', index = False)

print(stock_2_training_data)
print(stock_2_test_data)
print(stock_2_training_objective)
print(stock_2_test_objective)

all_training_data.to_csv('all_training.csv', index = False)
all_test_data.to_csv('all_test.csv', index = False)
all_training_objective.to_csv('all_trainingobjective.csv', index = False)
all_test_objective.to_csv('all_testobjective.csv', index = False)

print(all_training_data)
print(all_test_data)
print(all_training_objective)
print(all_test_objective)
```

Minimalfeaturestraining.py

We use the below to show with minimal features we can achieve 70.3% validation accuracy, showing the features we identified explain the data well.

```
    import tensorflow.keras as keras
import numpy as np
import pandas as pd

import keras_tuner as kt
from keras.models import Sequential
from keras.layers import Dense, Dropout, InputLayer
from keras.optimizers import Adam
from keras.callbacks import EarlyStopping
from keras.regularizers import l2
```

```
# Import the preprocessed data

new_train = pd.read_csv('all_training.csv')

# Select only the three features we found that appear to be useful
X_train = new_train.loc[:, ['price pattern chance', 'level 1 volume split',
'stock_label']]

Y_train = pd.read_csv('all_trainingobjective.csv')

new_val = pd.read_csv('all_test.csv')

# Select only the three features we found that appear to be useful
X_val = new_val.loc[:, ['price pattern chance', 'level 1 volume split',
'stock_label']]

Y_val = pd.read_csv('all_testobjective.csv')

print(X_train.head())
print(Y_train.head())
print(X_val.head())
print(Y_val.head())

print(X_train.shape[1])
input("tap enter to continue")
# Define the model with hyperparameters
def build_model(hp):

    # Choose the number of layers (this is the hyperparameter)
    num_layers = hp.Int('num_layers', min_value=1, max_value=10, step=1)
    # Tune between 1 and 10 layers

    model = Sequential()
    model.add(InputLayer(input_shape = (X_train.shape[1],)))
    units = hp.Int('units_1', min_value=32, max_value=128, step=32)
    model.add(Dense(units=units, kernel_regularizer=l2(0.01),
    activation=hp.Choice('activation1', ['relu'])))
    model.add(Dropout(rate=hp.Float('dropout1', 0.1, 0.5, step=0.1)))
    units = hp.Int('units_2', min_value=32, max_value=128, step=32)
    model.add(Dense(units=units, kernel_regularizer=l2(0.01),
    activation=hp.Choice('activation3', ['relu'])))
    model.add(Dropout(rate=hp.Float('dropout3', 0.1, 0.5, step=0.1)))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(optimizer=Adam(learning_rate=hp.Float('lr', min_value=1e-4,
    max_value=1e-2, sampling='LOG')),
                  loss='binary_crossentropy',
                  metrics=['accuracy'])
    return model
```

```
# Define the tuner
tuner = kt.Hyperband(build_model, objective='val_accuracy', max_epochs=10,
hyperband_iterations=2, directory='everything_Thomasmodel_big_layer',
project_name='hyperparameter_tuning')

early_stopping = EarlyStopping(monitor='val_loss', patience=5,
restore_best_weights=True)

# Search for the best hyperparameters
tuner.search(X_train, Y_train, epochs=10, validation_data=(X_val, Y_val),
callbacks=[early_stopping])

# Get the best model
best_model = tuner.get_best_models(num_models=1)[0]
best_hp = tuner.get_best_hyperparameters(num_trials=1)[0]

# Model achieves approximately 70.3% accuracy with about 10 minutes of searching
# for parameters.
```

References

[1] François Chollet et al. *Keras*. `https://keras.io`. 2015.

[2] Lukas Gonon. *Deep Learning Lecture Slides 2024*. Dec. 2024.

[3] Tom O'Malley et al. *KerasTuner*. `https://github.com/keras-team/keras-tuner`. 2019.

[4] Travis Oliphant, Eric Jones, and Pearu Peterson. *Scipy*. `https://scikit-learn.org/1.5/modules/preprocessing.html`. 2001.