# CNN Models for Predicting Price Direction from LOB data

Jack Thompson

January 2025

## Contents

# 1 Introduction

This document compares different variations of Convolutional Neural Network (CNN) models developed for predicting future price movements using Limit Order Book (LOB) data.

# 2 Data Preprocessing

## 2.1 1-Dimensional Data (Time Series)

The file `CNNdataprep.ipynb` is responsible for calculating, given a LOB file, a list of new metrics to track:

- Mid Price
- Spread
- Total Ask Volume
- Total Bid Volume
- Ask Volume up to Level 5
- Bid Volume up to Level 5
- Convert `Event_type` to a 1x5 binary array
- Order Book Imbalance (OBI)
- Relative Spread (RS)
- Change in Ask Price Level 1 (`Delta_AP_1`)
- Change in Bid Price Level 1 (`Delta_BP_1`)
- Weighted Average Price (WAP)
- Returns (log returns)
- Volatility of Returns with windows 5, 10, 15
- Moving Average of Returns with windows 5, 10, 15
- Moving Average of summed Order Direction 5, 10, 15

**Potential target variables:**

- Change in next Mid-Price (future)
- Next Mid Direction (future)

Excluding the target variables, and including the data provided in the original LOB snapshot, we are left with 69 different input channels for our model. Additionally, I opted to remove the first and last 30 minutes of trading data to avoid training the model on the market environment during these periods. Once saved as `fulldata.csv` (located in `/CNNdata`), this dataset serves as the default input for processing functions for each model.

## 2.2 2-Dimensional Data (Images)

The 2D images used to feed into the 2D CNNs are processed in the `CNNimagegen.py` file. The primary function is titled:

```
def data_frame_organise()
```

This function processes raw LOB data, retaining only numerical and relevant features, such as:

- Mid Price
- Spread
- Return (log return)
- Relative Ask Price Level 1,...,10
- Relative Bid Price Level 1,...,10
- Log Ask Volume Level 1,...,10
- Log Bid Volume Level 1,...,10

The processed data is then resampled into 100ms batches to capture meaningful mid-price fluctuations.

# 3 Image Creation

## 3.1 Single-Channel 2D Images

### 3.1.1 Image type 1

`create_image_updown_vol`: This function generates greyscale images where the right side represents LOB ask volume and the left side represents bid volume. Each LOB level spans from 1 (center) to 10 (edges), centered on the mid-price.

### 3.1.2 Image type 2

`create_image_updown_vol_mid`: A variation that removes the central separating line, replacing it with a single white dot.

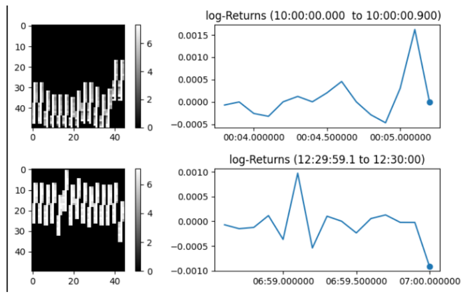


Figure 1: Single-Channel 2D Image Example

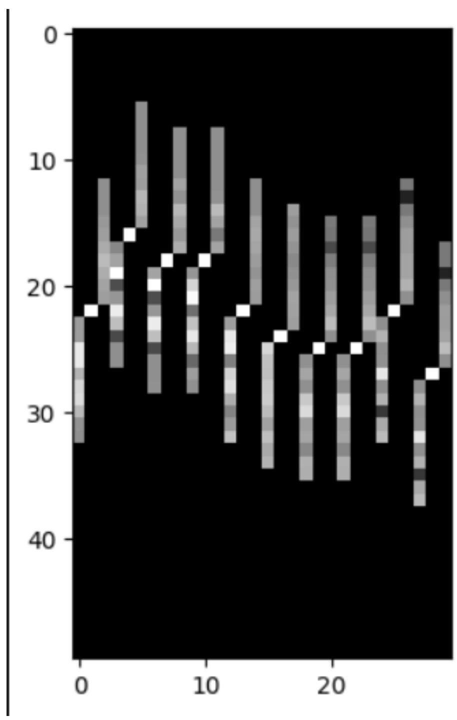

Figure 2: Single-Channel 2D Image Example with Mid-Price Dot

### 3.2 Triple-Channel 2D Images

In this structure, three unique non-overlapping channels are used:

- **Red Channel:** Ask data
- **Blue Channel:** Bid data
- **Green/Cyan Channel:** Mid-price

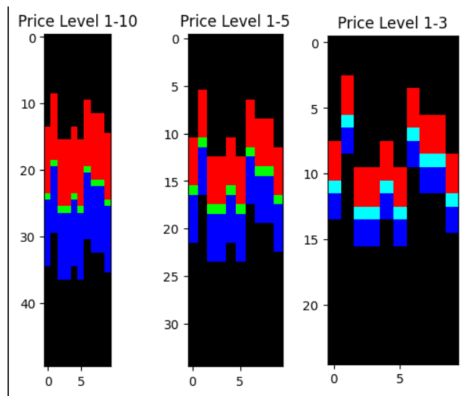


Figure 3: Triple Channel Images wit levels: 10,5,3

```
def create_image_updown_vol_mid_triple_l10()
def create_image_updown_vol_mid_triple_l5()
def create_image_updown_vol_mid_triple_l3()
```

# 4 CNN Architecture

Throughout the three different CNNs designed for my models, each one will be using Sparse-Categorical-Loss as a the loss function. As the target and output of all models will be either a 0,1,2 i.e integer categories.

## 4.1 2D CNN

### 4.1.1 Single Channel

The architecture processes the image_dimension[0] $\times$ image_dimension[1], LOB data as a 2D input, capturing patterns and relationships within the data. A convolutional layer with 32 filters and a horizontal kernel $(1 \times 3)$ extracts local features, followed by max pooling to reduce width and focus on key information. A second convolutional layer with 64 filters deepens the feature extraction, with another pooling layer to further compress the data.

The flattened output is passed through two dense layers: one with 128 neurons to learn complex patterns, and a final layer with 3 neurons using softmax activation to classify the data into buy, sell, or hold decisions. This design balances simplicity and effectiveness for the LOB data.

### 4.1.2 Triple Channel

The triple-channel architecture is designed to process three separate data streams: Bid, Ask, and Mid prices. Each channel corresponds to specific price information, isolating and focusing on unique patterns. The input is split into three distinct channels enabling independent processing of the Bid (Blue), Ask (Red), and Mid (Green) data.

Each channel undergoes individual convolutional layers, tailored to extract features at different resolutions. The Mid channel uses $5 \times 5$ kernels for broader feature extraction, while the Bid and Ask channels leverage $3 \times 3$ kernels for finer details. Max pooling follows each convolutional layer, reducing dimensionality and emphasizing significant features. After two convolutional-pooling stages per channel, the outputs are concatenated to integrate the extracted features.

The combined feature map is flattened and passed through two dense layers: one with 128 neurons to capture complex relationships and a final softmax layer with three outputs for classification. This architecture ensures a comprehensive analysis of the Bid, Ask, and Mid data streams, enabling robust decision-making.

## 4.2 1D CNN

The 1D architecture processes sequential data in a single dimension, ideal for time-series or similar structured inputs. The input shape is defined explicitly to establish the structure of the data.

The model begins with a 1D convolutional layer using 3-unit kernels to extract local patterns within the sequence. This is followed by Batch Normalization, which stabilizes and accelerates the training process by normalizing the activations. A MaxPooling layer reduces the dimensionality by pooling over two units, preserving key features while reducing computational complexity.

A second convolutional layer, identical to the first, further refines the feature extraction. Batch Normalization and MaxPooling are applied again to ensure stable learning and focus on essential patterns.

The output is flattened to convert the extracted features into a fully connected format. A Dense layer with 128 neurons captures higher-order relationships, and Dropout is employed to prevent overfitting. Finally, the softmax output layer classifies the data into three categories, completing the architecture.

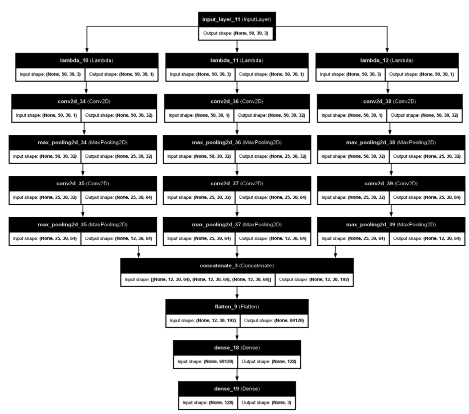

Figure 4: Triple Channel CNN Arhictecture for 2d Images

# 5 Comparison of Models

## 5.1 Distribution of Returns

At each time sampling the distribution of returns are slightly different due to the sampling process.

They can be seen in the images below.

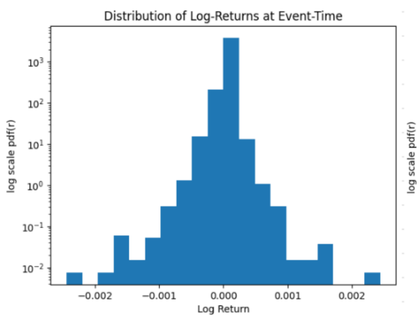


Figure 5: Event-Time

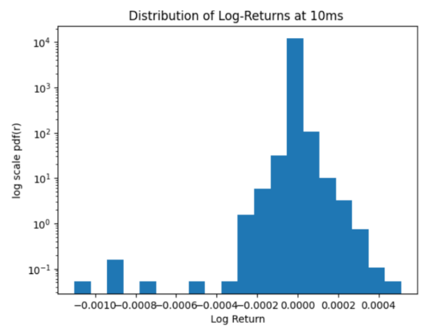


Figure 6: 10ms Sampling

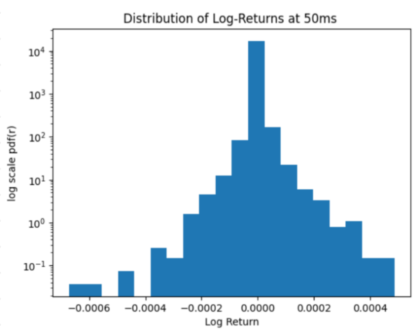


Figure 7: 50ms Sampling

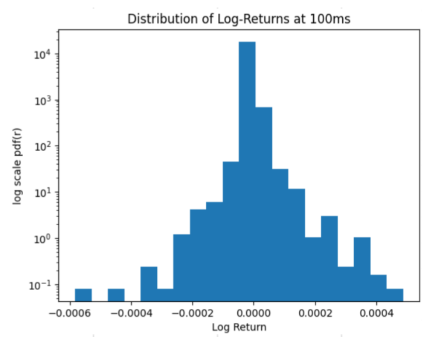


Figure 8: 100ms Sampling

## 5.2 Support Distributions

The supports (true labels) for different time sampling, and horizons are unique, and are recorded in the table below.

| Time Sample | Lookback Window | Prediction Horizon | Up (count) | Neutral (count) | Down (count) |
|---|---|---|---|---|---|
| Event Time | 15 | 1 | 4703 | 72501 | 4694 |
| Event Time | 15 | 10 | 21810 | 38887 | 21200 |
| 50ms | 5 | 1 | 5611 | 62982 | 1602 |
| 50ms | 15 | 10 | 11283 | 47744 | 11156 |
| 100ms | 10 | 10 | 14698 | 10074 | 14826 |

| Time Sample | Lookback Window | Prediction Horizon | Up % | Neutral % | Down % |
|---|---|---|---|---|---|
| Event Time | 15 | 1 | 5.74% | 88.53% | 5.73% |
| Event Time | 15 | 10 | 26.63% | 47.48% | 25.89% |
| 50ms | 5 | 1 | 7.99% | 89.72% | 2.28% |
| 50ms | 15 | 10 | 16.08% | 68.03% | 15.90% |
| 100ms | 10 | 10 | 37.12% | 25.44% | 37.44% |

## 5.3 Model Results

| Model | Im Generator | TimeSampling | LB | Horizon | Im. Dimension | Batch Size | Epochs | Train Accuracy | Val. Accuracy | Train Loss | Val. Loss | Test Accuracy | W. F1-Score |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CNN2d | updown_vol_mid | 100ms | 10 | 1 | 1x50x30 | 64 | 3 | 62.00% | 58.20% | 0.85 | 0.89 | 58.20% | 0.55 |
| CNN2d | updown_vol | 100ms | 10 | 10 | 1x50x30 | 64 | 10 | 71.40% | 61.50% | 0.67 | 0.88 | 61.50% | 0.60 |
| CNN2d | updown_vol_mid_ | 100ms | 10 | 10 | 3x50x10 | 64 | 10 | 70.10% | 59% | 0.67 | 0.95 | 60.10% | 0.57 |
| CNN2d | updown_vol_mid_ | 100ms | 10 | 10 | 3x35x10 | 64 | 10 | 74.70% | 52.50% | 0.59 | 1.25 | 53.10% | 0.52 |
| CNN2d | updown_vol_mid_ | 100ms | 10 | 10 | 3x25x30 | 64 | 10 | 75.40% | 49.70% | 0.57 | 1.42 | 50.10% | 0.47 |
| CNN1d | n/a | Event Time | 15 | 1 | 69x1 | 256 | 5 | 97.70% | 18.00% | 0.07 | 4.16 | 8.90% | 0.07 |
| CNN1d | n/a | Event Time | 15 | 10 | 69x1 | 32 | 5 | 91.30% | 73.00% | 0.23 | 0.69 | 76% | 0.76 |
| CNN1d | n/a | 50ms | 15 | 10 | 69x1 | 32 | 5 | 91.50% | 92.70% | 0.23 | 0.22 | 91% | 0.91 |
| CNN1d | n/a | 100ms | 10 | 1 | 69x1 | 364 | 100 | 80.50% | 78.80% | 0.21 | 0.39 | 89.50% | 0.87 |
| CNN1d | n/a | 50ms | 5 | 1 | 69x1 | 32 | 50 | 82.90% | 80.00% | 0.17 | 0.53 | 82% | 0.87 |

Figure 9: Model Results

It is evident that the strongest models in general were designed for 1D inputs.

I am going to title the strongest performing model, with the F1-score of 0.91: **C150**.

# 6 Further Testing

Taking **C150** and running it for more epochs.

After 15 epochs:

accuracy: 0.9457 - loss: 0.1515 - validation accuracy: 0.9456 - validation loss: 0.1526

After 30 epochs:

accuracy: 0.9613 - loss: 0.1094 - validation accuracy: 0.9444 - validation loss: 0.177

Although there was a training metric improvement, there was minimal change in the validation testing on unseen data. Therefore, to risk over-fitting, this is as far as we will train the model.

### 6.1 Changing Stock

So far we have been runnning the models and training them on Tesla (TSLA) LOB data.

Now let's introduce Facebook (FB) LOB data.

First just running the TSLA trained model to test on FB data:

using the same preproccessing steps, and fitting the X values of the metrics of FB to the same scaler that was fit on the TSLA values.

```
FB Accuracy: 96.18%
14625/14625 ━━━━━━━━━━━━━━━━━━━━ 38s 3ms/step
FB Test Accuracy (using predictions): 0.9618160099664296
              precision    recall  f1-score   support

         0.0       0.95      0.90      0.92     75567
         1.0       0.99      0.98      0.98    319136
         2.0       0.87      0.95      0.91     73268

    accuracy                           0.96    467971
   macro avg       0.94      0.94      0.94    467971
weighted avg       0.96      0.96      0.96    467971
```

Figure 10: C150 running on FB data

## 7 Conclusion

This reinforces the idea that the model has learned a generalised pattern in 1D LOB images. As it ran an even higher accuracy and F1 score on a dataset of a small-tick stock after being trained on a high-tick stock.

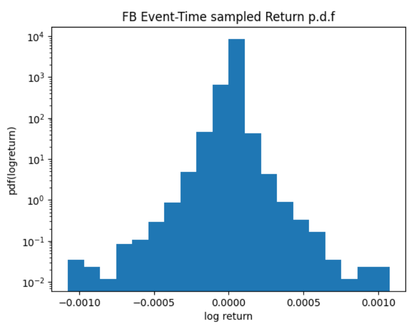


Figure 11: FB Event-Time Returns

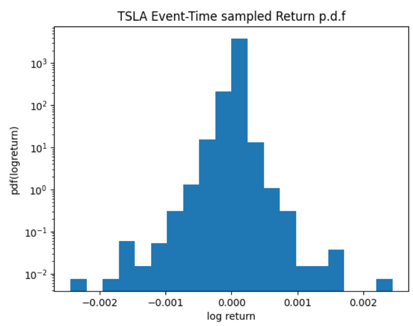


Figure 12: TSLA Even-Time Return

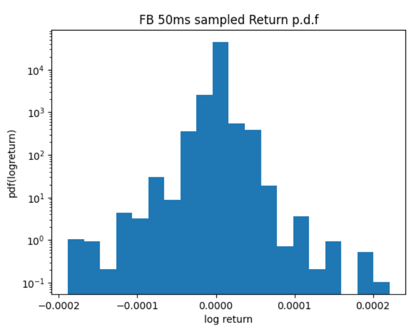


Figure 13: FB 50ms Sampled Returns

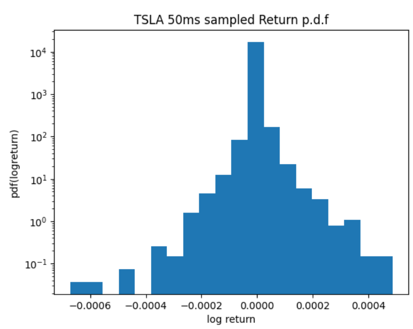


Figure 14: TSLA 50ms Sampled Returns

It is also further clear that out of all variations I tested the properties of the **C150** model is, or is near, optimal: 50ms time sampling, batch size 32, epochs 5-15.

### 7.1 Further Work

In order to solidify my findings, and be sure that the model **C150** 1-dimensional CNN is a reliable and accurate predictor for high frequency stock price movement in the next 10-events (0.5 seconds), I need to do further simulations on current data, and then backtesting to see if I can leverage the model predictions to make a investment strategy with a high Sharpe Ratio.

But due to my computational and resource constraints, this is not currently possible. As well as, I will no longer have much time to dedicate to this as I resume my studies. This was a project to distract me over my Christmas break, which produced very promising results.

## A Appendix

All code, notebooks, cleaned datasets and testing programmes can be found on the GitHub repo: `https://github.com/jackthompsondb/CNN_HighFreqLOB`