CNN Models for Predicting Price Direction from LOB data

Jack Thompson

January 2025

Abstract

This paper investigates the use of Convolutional Neural Networks (CNNs) for predicting future price movements using Limit Order Book (LOB) data. I developed and compared several CNN architectures, including 1D and 2D models, to evaluate their effectiveness in capturing complex patterns within high-frequency trading data. The preprocessing pipeline involved transforming raw LOB data into both time-series and image formats, enabling diverse model inputs.

The results indicate that the 1D CNN model, named **C150**, achieved the highest performance with an F1-score of 0.91, demonstrating robust predictive capabilities across different stocks. Further testing on unseen data confirmed the model's generalization ability. Future work will focus on optimizing the model for real-time trading strategies and exploring its application across various market conditions. This study contributes to the growing field of AI-driven financial forecasting, highlighting the potential of CNNs in high-frequency trading environments.

Contents

Note:This project was undertaken independently over the Christmas break, serving as both a learning experience and a personal exploration into the application of CNNs for financial forecasting. While the results are promising, it's important to note that this work is not at a professional standard and further refinement and validation are needed.

1 Introduction

The current state of financial markets are creating heavy interest around the development of sophisticated models capable of predicting price movements with high accuracy at high frequency levels. Limit Order Book (LOB) data, which provides a detailed view of market dynamics, has emerged as a valuable resource for such predictive modeling. Convolutional Neural Networks (CNNs), known for their ability to capture spatial hierarchies in data, offer a promising approach to analyzing LOB data.

This study explores the application of CNNs to predict stock price directions by leveraging both 1D and 2D (and multi-channel) representations of LOB data. By transforming raw LOB data into structured inputs, the research aims to uncover patterns that traditional methods might overlook. The focus is on evaluating the effectiveness of different CNN architectures in capturing these patterns and their potential to generalize across various market conditions.

Through rigorous experimentation, this work seeks to contribute to the field of AI-driven financial forecasting, providing insights into the design and implementation of CNN models for high-frequency trading environments.

2 Existing Work

My research was inspired by several key studies in the field of stock price trend prediction using Limit Order Book (LOB) data. The paper by Matteo Prata et al. [1] titled "LOB-Based Deep Learning Models for Stock Price Trend Prediction: A Benchmark Study" source played a pivotal role in sparking my interest in the application of Convolutional Neural Networks (CNNs) for this purpose. This study provided a comprehensive benchmark of deep learning models, highlighting the potential of CNNs in capturing intricate patterns within LOB data.

Additionally, the work by Wuyi Ye, Jinting Yang, Pengzhan Chen [2] titled "Short-term Stock Price Trend Prediction with Imaging High Frequency Limit Order Book Data" source inspired the creation of 2D images from LOB data. This approach led me to explore and extend the concept from single-channel to triple-channel imaging, although my most successful results were ultimately achieved using 1D imaging techniques.

For the LOB data utilized in my research, I relied on the LOBSTER data resource, which provided the high-frequency data necessary for training and testing my models.

3 Data Preprocessing

3.1 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.

3.2 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.

4 Image Creation

4.1 Single-Channel 2D Images

4.1.1 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.

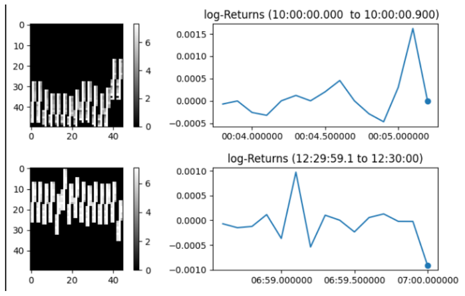


Figure 1: Single-Channel 2D Image Example

4.1.2 Image type 2

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

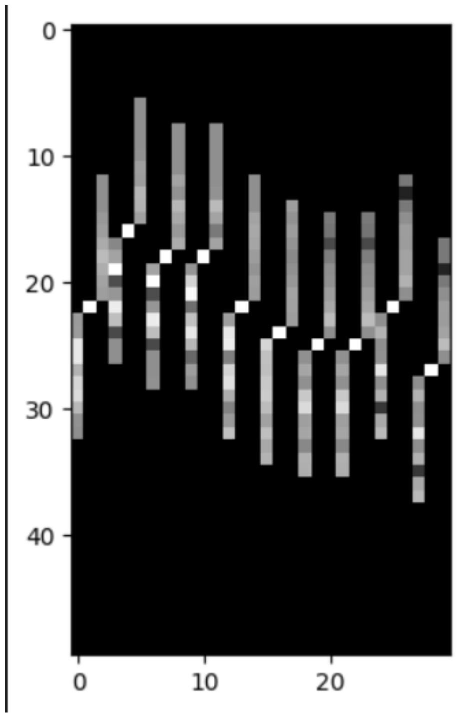


Figure 2: Single-Channel 2D Image Example with Mid-Price Dot

4.2 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

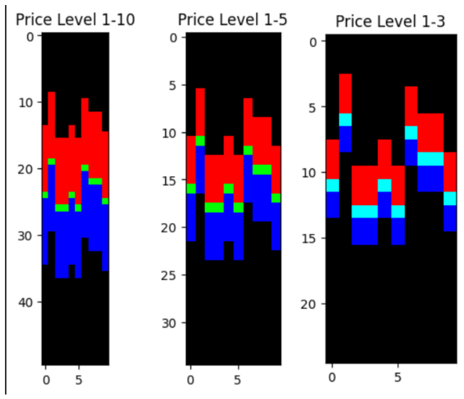


Figure 3: Triple Channel Images wit levels: 10,5,3

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

5 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.

5.1 2D CNN

5.1.1 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 (1×3) 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.

5.1.2 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 5×5 kernels for broader feature extraction, while the Bid and Ask channels leverage 3×3 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.

5.2 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.

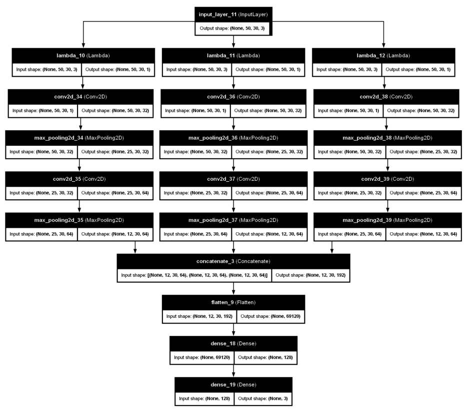

input_layer_11 (InputLayer)
Output shape: (None, 50, 30, 3)
lambda_10 (Lambda)
Input shape: (None, 50, 30, 3)
Output shape: (None, 50, 30, 1)
lambda_11 (Lambda)
Input shape: (None, 50, 30, 3)
Output shape: (None, 50, 30, 1)
lambda_12 (Lambda)
Input shape: (None, 50, 30, 3)
Output shape: (None, 50, 30, 1)
conv2d_34 (Conv2D)
Input shape: (None, 50, 30, 1)
Output shape: (None, 50, 30, 32)
conv2d_36 (Conv2D)
Input shape: (None, 50, 30, 1)
Output shape: (None, 50, 30, 32)
conv2d_38 (Conv2D)
Input shape: (None, 50, 30, 1)
Output shape: (None, 50, 30, 32)
max_pooling2d_34 (MaxPooling2D)
Input shape: (None, 50, 30, 32)
Output shape: (None, 25, 30, 32)
max_pooling2d_36 (MaxPooling2D)
Input shape: (None, 50, 30, 32)
Output shape: (None, 25, 30, 32)
max_pooling2d_38 (MaxPooling2D)
Input shape: (None, 50, 30, 32)
Output shape: (None, 25, 30, 32)
conv2d_35 (Conv2D)
Input shape: (None, 25, 30, 32)
Output shape: (None, 25, 30, 64)
conv2d_37 (Conv2D)
Input shape: (None, 25, 30, 32)
Output shape: (None, 25, 30, 64)
conv2d_39 (Conv2D)
Input shape: (None, 25, 30, 32)
Output shape: (None, 25, 30, 64)
max_pooling2d_35 (MaxPooling2D)
Input shape: (None, 25, 30, 64)
Output shape: (None, 12, 30, 64)
max_pooling2d_37 (MaxPooling2D)
Input shape: (None, 25, 30, 64)
Output shape: (None, 12, 30, 64)
max_pooling2d_39 (MaxPooling2D)
Input shape: (None, 25, 30, 64)
Output shape: (None, 12, 30, 64)
concatenate_3 (Concatenate)
Input shape: [(None, 12, 30, 64), (None, 12, 30, 64), (None, 12, 30, 64)]
Output shape: (None, 12, 30, 192)
flatten_9 (Flatten)
Input shape: (None, 12, 30, 192)
Output shape: (None, 69120)
dense_18 (Dense)
Input shape: (None, 69120)
Output shape: (None, 128)
dense_19 (Dense)
Input shape: (None, 128)
Output shape: (None, 3)

Figure 4: Triple Channel CNN Arhictecture for 2d Images

6 Comparison of Models

6.1 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.

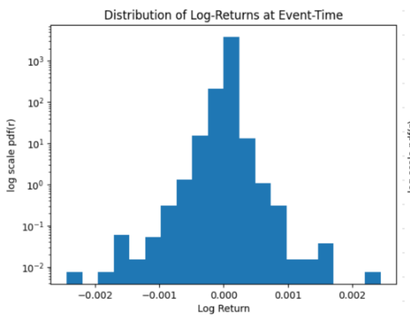


Figure 5: Event-Time

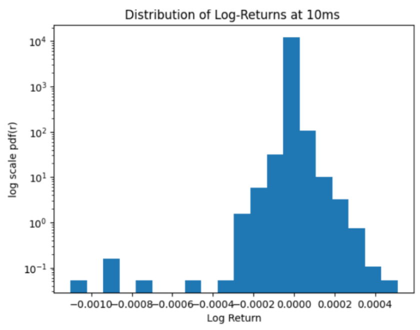


Figure 6: 10ms Sampling

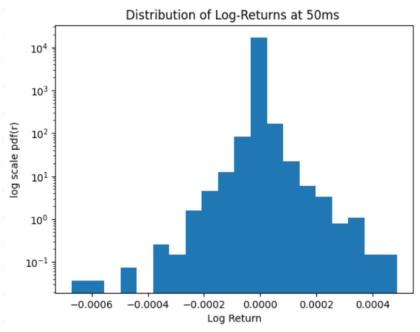


Figure 7: 50ms Sampling

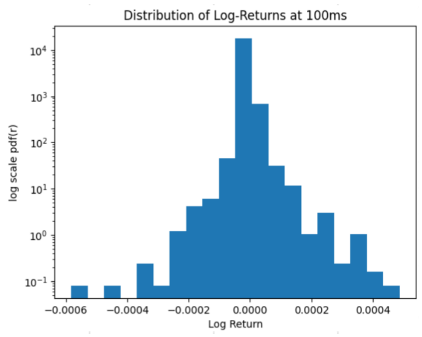


Figure 8: 100ms Sampling

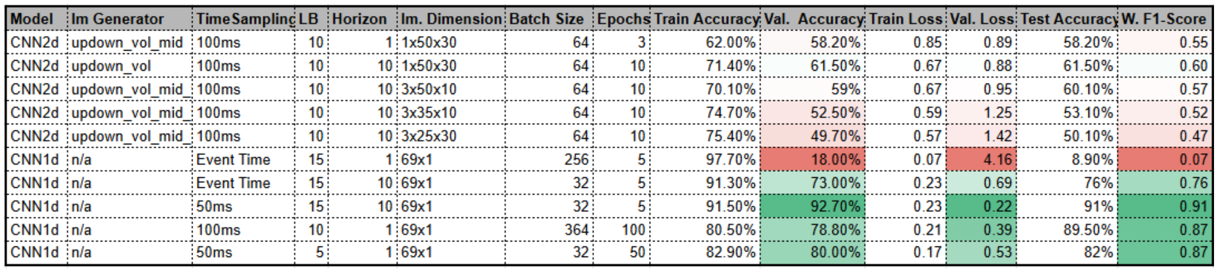
6.2 Support Distributions

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%

6.3 Model Results



Model	Im Generator	TimeSamplinc	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

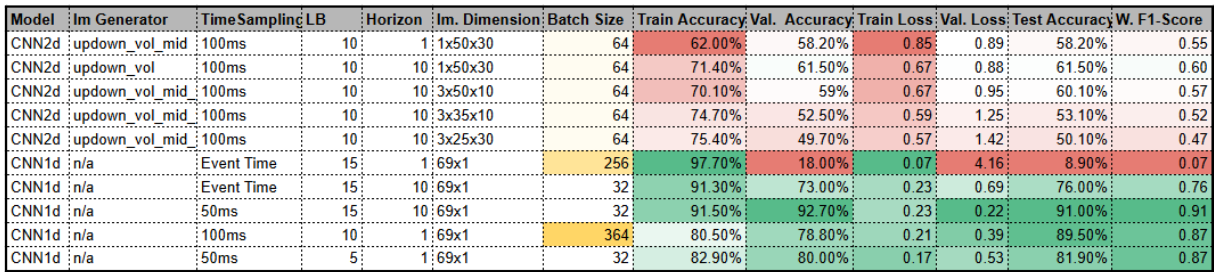
Figure 9: Model Results

The most important performance metrics have been conditionally formatted. As we can see, they all perfectly correlate with each other in terms of ranking.

Worst performing in general seemed to be the 2D imaging models. Out of the 2D imaging models, the triple-channel with price-level 3, and both single-channel images did fairly similar, but not brilliant.

By a clear divide, the networks designed to take in the 1D data, outperformed the 2D, except one outlier which we mention after figure 10.

Now this could have been a result due to my own fault. It is possible my architecture for the 2D imaging was not sufficient, and there could be much room for improvement. It is a far more complicated process to refine.



Model	Im Generator	TimeSamplinc	LB	Horizon	Im. Dimension	Batch Size	Train Accuracy	Val. Accuracy	Train Loss	Val. Loss	Test Accuracy	W. F1-Score
CNN2d	updown_vol_mid	100ms	10	1	1x50x30	64	62.00%	58.20%	0.85	0.89	58.20%	0.55
CNN2d	updown_vol	100ms	10	10	1x50x30	64	71.40%	61.50%	0.67	0.88	61.50%	0.60
CNN2d	updown_vol_mid_	100ms	10	10	3x50x10	64	70.10%	59%	0.67	0.95	60.10%	0.57
CNN2d	updown_vol_mid_	100ms	10	10	3x35x10	64	74.70%	52.50%	0.59	1.25	53.10%	0.52
CNN2d	updown_vol_mid_	100ms	10	10	3x25x30	64	75.40%	49.70%	0.57	1.42	50.10%	0.47
CNN1d	n/a	Event Time	15	1	69x1	256	97.70%	18.00%	0.07	4.16	8.90%	0.07
CNN1d	n/a	Event Time	15	10	69x1	32	91.30%	73.00%	0.23	0.69	76.00%	0.76
CNN1d	n/a	50ms	15	10	69x1	32	91.50%	92.70%	0.23	0.22	91.00%	0.91
CNN1d	n/a	100ms	10	1	69x1	364	80.50%	78.80%	0.21	0.39	89.50%	0.87
CNN1d	n/a	50ms	5	1	69x1	32	82.90%	80.00%	0.17	0.53	81.90%	0.87

Figure 10: More Highlighted Results

We can highlight the rest of the metrics to make clearer some other relationships in the data (figure 10).

The large batch size seems to be a likely cause of over-fitting. We can see the 256 batch size model scored the highest Training Accuracy and Training Loss, but simultaneously scored the lowest Validation Accuracy, Validation Loss, and Test Accuracy. The model was too over-fit on our training data.

However, we do have a batch size of 364 in one model, the difference between this and the 256 is that here we are running in 100ms samples and lookback window of 10 in instead of event-time. And surprisingly, this model was pretty strong. In fact, it scored tied 3rd/4th place overall.

I am naming the clearly strongest performing model, with the F1-score of 0.91: **C150**.

And this model is what we are going to take away from our mass testing, and we see if it is an appropriate model for general, real-world use, through further testing.

7 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.

7.1 Inference Time

Predicting ahead 10 time-steps, at a 50ms sampling rate, means that the model is predicting the state of the market 500ms, or 0.5s ahead. Thus, it is important

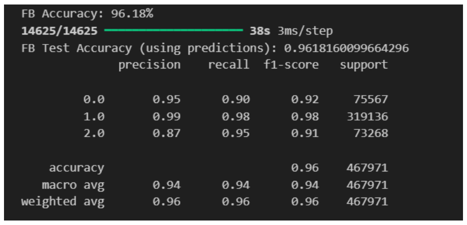
that the inference time is far less than this, since in a real-world scenario there would be many more time barriers if a model were to be implemented and run on live data.

To calculate inference time, I split up my training data into a batch size of 32, as **C150** uses. And timed the inference time for each batch and then took an average. This was over the entire dataset which is approximately greater than 500,000 rows long originally.

Our end results yields an average inference time of: **65ms** or **0.065s**

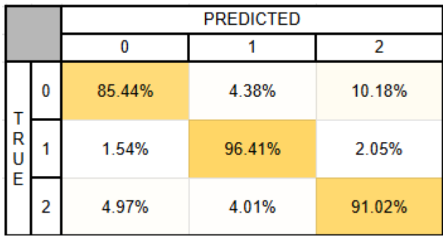
7.2 Testing on Unfamilair Data

So far we have been runnning the models and training them on Tesla (TSLA) LOB data. Now let's introduce Facebook (FB) LOB 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.



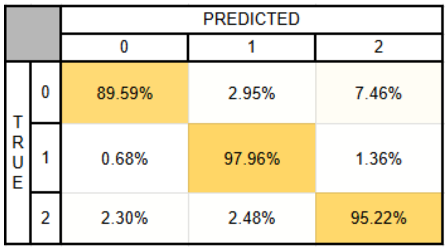
```
FB Accuracy: 96.18%
14625/14625 ━━━━━━━━━━━━━━━━━━━━ 38s 3ms/step
FB Test Accuracy (using predictions): 0.9618160099664296
              precision    recall  f1-score   support

         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 11: **C150** running on FB data



		PREDICTED		
		0	1	2
TRUE	0	85.44%	4.38%	10.18%
	1	1.54%	96.41%	2.05%
	2	4.97%	4.01%	91.02%

Figure 12: Confusion Matrix When Tested on TSLA data



		PREDICTED		
		0	1	2
TRUE	0	89.59%	2.95%	7.46%
	1	0.68%	97.96%	1.36%
	2	2.30%	2.48%	95.22%

Figure 13: Confusion Matrix When Tested on FB data

8 Conclusion

This result underscores the model's (**C150**) ability to generalize patterns in 1D LOB images, as evidenced by its improved accuracy and F1 score when applied to a small-tick stock dataset, despite being trained on a large-tick stock.

Due to how flawless the model transitioned from a one completely different stock and LOB environment to another, after being trained on a single set, gives me confidence that future research and use of this particular model will yield equally promising results.

The result of 1-dimensional time series data outperforming 2-dimensional images I will put down to the rule of simplicity. There is a good chance that if done correctly, and perfected, that a 2-dimensional and multi-channel image input for a CNN would outperform and make more reliable predictions, but there are exponentially many ways to order/arrange the data and then design architecture for that it is beyond the scope of my time and expertise.

8.1 Future Steps

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 happened to produce very promising results.

A Appendix

All code such as imaging functions, model training and anlysis notebooks, the C150 model itself, and more can be found on the GitHub repository: `https://github.com/jackthompsondb/CNN_HighFreqLOB`

References

[1] [Matteo Prata, Giuseppe Masi, Leonardo Berti, Viviana Arrigoni, Andrea Coletta, Irene Cannistraci, Svitlana Vyetrenko, Paola Velardi, Novella Bartolini] *LOB-Based Deep Learning Models for Stock Price Trend Prediction: A Benchmark Study*, `https://arxiv.org/pdf/2308.01915`.

[2] [Wuyi Ye, Jinting Yang, Pengzhan Chen], *Short-term Stock Price Trend Prediction with Imaging High Frequency Limit Order Book Data*, `https://doi.org/10.1016/j.ijforecast.2023.10.008`.