Norwegian School of Economics
Bergen, Autumn 2024

Deep Learning-Based Stock Price Prediction for Norwegian Companies

A Comparative Study

Maysam Rashidi Ranjbar, Moein Sahraei

Supervisor: Professor Walter Pohl

Master thesis, Economics and Business Administration Major: Business Analytics and Finance

NORWEGIAN SCHOOL OF ECONOMICS

This thesis was written as a part of the Master of Science in Economics and Business Administration at NHH. Please note that neither the institution nor the examiners are responsible − through the approval of this thesis − for the theories and methods used, or results and conclusions drawn in this work.

Abstract

This thesis discusses the applicability of advanced machine learning (specifically GBDT models) and deep learning models in financial forecasting, especially in the setting of stock price prediction on the TITLON dataset. With this representation of the Oslo Stock Exchange data, which has complexities and volatilities intrinsic to financial markets, prediction of the stock prices is quite challenging. That would mean developing and evaluating robust models that are able to capture intricate patterns in data.

The research is done at a holistic level: the performance of many models of machine learning is estimated, including Gradient Boosting Decision Trees like XGBoost and CatBoost, alongside some deep learning state-of-the-art architectures like Self-Normalizing Networks, Grownnet, DCNv2, AutoInt, MLP, ResNet, and FT-Transformer. These models will be contrasted for their capacity to reduce the error in predictions; two metrics we use to assess the performance of models are the root mean squared error and the mean absolute error.

Another important novelty of this research is the application of a feature extraction with CNN for improving the model toward its precision in accuracy. Such results, before and after applying the CNN feature extraction, are rigorously compared to show that most of the models—deep learning architectures like SNN, DCNv2, AutoInt, and MLP—are substantially improved by this method. These models depicted a large reduction in RMSE and MAE, which goes in line with improved predictive accuracy and generalization ability to unseen data (the test set).

Models such as ResNet and FT-Transformer demonstrated smaller improvements after feature extraction, proving that model-specific tuning could be required to some extent. GBDT models had massive performance improvements, particularly on the MAE, hence proving their robustness and competitiveness toward financial forecasting tasks when matched with efficient features.

This thesis therefore concludes that although CNN-based feature extraction has the capacity to increase a model's predictive performance significantly, the choice of the appropriate model architecture and its hyperparameter tuning remain a prerequisite for optimal results. The findings enrich the literature on deep learning and machine learning techniques applied to financial forecasting, including valuable insights for future studies and practical implementations within the financial industry.

Contents

Contents

List of figures

List of Tables

1. Introduction

According to Sarker, I.H (2021) "Deep learning is a subset of machine learning that involves neural networks with multiple layers (hence 'deep') that can learn from large amounts of data. It mimics the human brain's ability to recognize patterns and make decisions" (Sarker, I.H, 2021). The origin of the deep learning models is artificial neural network (ANN), that includes connected processing units called neurons. In a deep learning model, each instance of the input data goes through multiple layers of transformations by adding a bias, (*b*) getting multiplied by a weight (*w*) and then goes into the neuron (Sarker, I.H, 2021). Each neuron has its own activation function that transforms the input and sends the transformed input to the next layer. Weights and biases are determined by using “Backpropagation” type algorithms.

Recent studies by Gorishniy, Rubachev, Khrulkov, and Babenko (2021) studied the effectiveness of deep learning models for tabular data with focus on two main families of deep learning architectures. The first one is ResNet like architecture and the second one is Transformer architecture. They also compared these architectures to GBDT models and at the end concluded that there is no universally superior model. In this thesis, we decided to use models and architectures that are mentioned in this paper for several reasons. First, the study is focused on tabular data which encompasses stock market data. Second, it covers two of the most powerful deep learning structures and two of the most popular GBDT models so we believe that using these models and comparisons can show the effectiveness of both deep learning models and GBDT models in predicting stock prices of Norwegian companies.

Beside deep learning models, Gradient Boosted Decision Trees (GBDT) models are also among the most popular tools to predict stock prices. GBDT models are trained with a powerful technique called “ensemble learning” which refers to an iterative process of initializing the model with just one tree, calculating the error of the model, fit a new tree to the residuals, adding the predictions of this model to the previous one, specifying a learning rate that determines the contribution of each tree in the final prediction and keep doing this process until the stopping criterion is achieved (for example n number of trees are fitted). This process gradually improves the model performance because each tree is fixing the errors of the previous ones and combining them together can make an extremely powerful model (Hastie, Tibshirani, and Friedman 245). In this thesis we use two GBDT models

called XGBoost and CatBoost and compare their performance to deep learning models. GBDT models have some advantages that make them preferable to deep learning models. For example, they do not have the "black box" problem and are often much more interpretable than deep learning models. If some researcher is interested in the effects of different factors on the stock price, deep learning models do not have much to offer but GBDT models can yield useful information regarding variable importance. They also require less data to perform well compared to deep learning models. Furthermore, they require less computational power for training which is one of the main weaknesses of the deep learning models. Therefore, if in any scenario GBDT models offer the same performance as the deep learning models, they are always preferred (Chen & Guestrin, 2016).

Oslo Stock Exchange or Oslo Børs is the Norway's only stock exchange market that provides range of financial products such as equities, derivatives and other financial instruments. Since June 2019, the Euronext Group controls the Oslo Stock Exchange (Euronext website). TITLON is a dataset provided by UiT Noregs arktiske universitet that includes financial data of the Oslo Stock Exchange from 1980. This includes stocks, indices, bonds, funds and derivatives (UiT website). In this thesis, we are only focused on stocks to keep the study focused and analyze all the important details.

Stock trading is a very competitive and challenging task. Therefore, companies and individuals who wish to profit from the market must always keep up with the new technology and new methods of stock prediction. In recent years, utilizing deep learning models for stock price prediction has been very popular and profitable for both individuals and firms (Nelson, Pereira, & de Oliveira, 2017). The primary attractions of deep learning models for stock market traders are their ability to analyze a vast amount of complex and high dimensional data, learning from the data and generating valuable and precise predictions (Nelson, Pereira, & de Oliveira, 2017). However, proper implementation of deep learning in finance has proven to be an extremely challenging task which stems from the complexity and the unpredictability of the market. This unpredictability is mainly due to the phenomenon known as the "curse of dimensionality" which refers to the "exponential increase in volume associated with adding extra dimensions to Euclidean space, making high-dimensional data analysis complex" (Keogh & Mueen, 2017).

The curse of dimensionality in the stock market simply means that there are an extremely high number of factors that affect stock prices, and this makes the stock market a very

complex system. Therefore, every factor can exponentially add to this complexity. Deep learning models are one of the popular tools that stock market traders use to handle this complexity effectively because deep learning models are themselves very complex. That is why sometimes they are referred to as “black box” because the processing layers of neurons can get so complicated that human minds cannot comprehend the whole process that is being implemented on the input data (Tishby & Zaslavsky, 2015).

But properly implementing deep learning models is not without challenges. One of the challenges is the fact that they lack interpretability and work as a “black box” (Lipton, 2016) as mentioned before, they need powerful computational resources that can be extremely expensive, they have multiple hyperparameters such as learning rate, batch size and network architecture that must be tuned multiple times and this can add to their cost because hyperparameter tuning is inherently computationally expensive. They also tend to overfit the data if the training data does not represent the real-world phenomenon. And there are also some ethical problems that is beyond the scope of this research. There challenges are daily problems for companies and people who wish to utilize various deep learning methods for stock market predictions, some of them like overfitting have solutions, and others like computational expensiveness does not have any cure yet (Goodfellow et al., 2016).

In this study, feature extraction plays a critical role in enhancing the predictive accuracy of stock price models. We employ a Convolutional Neural Network (CNN) to extract short-term spatial features from financial time series data. This approach allows the model to capture complex, non-linear relationships within the data, which are often missed by traditional methods. Principal Component Analysis (PCA) is then utilized to reduce the dimensionality of the extracted features, ensuring that the most significant information is retained for predictive modeling.

The remaining of the thesis continues as follows, Chapter 2 describes the dataset in detail, including interesting patterns, correlations, statistical summary and other exploratory data analysis methods. Chapter 3 explains each model we use in our comparative study, it covers both deep learning models and GBDT models. Chapter 4 shows the methodology for transforming data into tensors using CNN. Chapter 5 presents the results of each model in detail with interpretation of numbers and charts and the conclusion.

2. Data

2.1 Descriptive analysis

The data we use to train and evaluate models is TITLON dataset provided by 'UiT the Arctic University of Norway'. We use the closing price as the response variable and our predictors are Fama-French factors : SMB (Small Minus Big) , HML (High Minus Low) and MOM (Momentum) , LIQ Pastor-Stambaugh factor (Liquidity Pastor-Stambaugh Factor), ,Volume and log return of the adjusted price, the moving average convergence divergence value MACD (Moving Average Convergence Divergence) , WPR (Williams %R), the signal line of MACD, volatility, moving averages (MA5, MA10, MA20), BIAS indicators (BIAS5, BIAS10), Relative Strength Index RSI for 6 and 12 periods and stochastic oscillators (%K and %D). The dataset contains 1,336,820 rows that covers 625 Norwegian companies.

Our feature selection for financial forecasting in this thesis is strategic, informed by well-articulated theories and best practices in quantitative finance. We conclude with the closing price, arguably the most basic indicator on which technical analysis is hinged. It's the very bedrock upon which key financial metrics-return and trend-are calculated; thus, it forms the bedrock basis of market understanding. We consider the Fama-French factors, including SMB and HML, which allow for capturing both the size and value effects in asset pricing. For example, SMB will reflect higher risk-adjusted returns associated with smaller companies, while HML will distinguish value from growth stocks. Further, we include the liquidity factor LIQ to be material, with liquidity itself as a significant determinant of stock price; generally, lower premiums against risk have been found on more liquid assets. The MOM factor controls for momentum in our model and captures a well-documented trend that assets which have been performing well in the past tend to continue performing well in the short term. We further use MA5, MA10, and MA20 to smooth out the price data and emphasize the movement of the trends over various time horizons that would help us in predicting market turning points. The Indicators of Bias-BIAS5, BIAS10-display prices deviated from moving averages and, therefore, overbought or oversold conditions to anticipate any correction in price. We use the Relative Strength Index (RSI6 and RSI12) as a tool to realize the speed and change of movement in prices, enabling us, accordingly, to make some judgments upon the turning point in the market for different time frames. Indicators of the Stochastic Oscillator-the Stoch_K and Stoch_D-indicate the relationship

between a security's closing prices and its price range over a period of time. These indicators are crucial in momentum-based trading strategies, as they allow the determination of extreme market conditions. MACD and Signal Line express the relationship between two moving averages, an important buy or sell indicator for the detection of changes in trends. Williams %R works like the Stochastic Oscillator in showing overbought/oversold conditions to facilitate decisions in extremes. We also assess volatility-both VOL1 and VOL2-in order to capture the magnitude of changes in prices, something important for the management of risk and anticipation of the behavior of markets during high moments of stress. Finally, Delta_MA5 captures the change in the 5-day moving average and provides a speedy evaluation of changing short-term market sentiment. Such a wide combination of features allows our model to represent dynamics of asset pricing, technical analysis indicators, risk and volatility management, trends in the short- and long-term perspective, while providing tools to detect anomalies in the market which might affect the pricing of stocks. The dataset is daily with the start date of 1997-06-02 and it continues to 2024-03-22, covering stock prices for both before and after Euronext Group takes control of the Oslo Stock Exchange. Here is the summary statistics of the most important variables in the dataset.

Feature	Count	Mean	Std	Min	25%	50% (Median)	75%	Max	Skew	Kurtosis
Close	1336820.0	72.8	176.03	-0.01	6.95	29.8	86.0	4900.0	11.53	184.19
SMB	1336820.0	0.0	0.01	-0.06	-0.0	0.0	0.01	0.34	2.89	49.68
HML	1336820.0	-0.0	0.03	-0.24	-0.01	-0.0	0.01	0.79	2.26	139.89
LIQ	1336820.0	0.01	0.02	-0.12	-0.01	-0.0	0.01	0.88	3.39	39.14
MOM	1336820.0	0.0	0.02	-0.05	-0.01	0.0	0.01	0.36	2.49	115.16
MA5	1336820.0	89.79	375.77	-0.0	31.39	61.04	106.7	5206.8	10.37	195.45
MA10	1336820.0	89.98	347.29	-0.0	40.14	65.2	104.45	3184.0	7.22	97.52
MA20	1336820.0	89.81	418.48	-0.0	46.0	68.05	100.54	1885.15	4.98	45.69
BIAS5	1336820.0	-1340000000000.0	45300000000000.0	-3800000000000000.0	-89.5	-51.75	65.75	124000.0	-49.43	2906.05
BIAS10	1336820.0	13600000000000.0	207000000000000.0	-514000000000000.0	-99.4	-56.32	42.27	2.36e+17	73.84	7438.95
RSI6	1336820.0	47388.0	527650.0	-145860.0	29.55	50.0	68.75	33905000.0	-221.38	58347.51
RSI12	1336820.0	49590.0	19799.0	-3.67	36.46	50.0	62.79	100.0	-0.17	-0.17
Stoch_K	1336820.0	49149.0	87124.0	0.0	22.86	50.0	75.86	11429000.0	-1.2	-1.2
Stoch_D	1336820.0	49524.0	87287.0	0.0	25.8	49.07	73.33	100.0	-1.19	-1.19
MACD	1336820.0	-0.01	0.03	-0.92	-1.88	-0.05	1.62	1.05	-1.57	40.71
Signal	1336820.0	-0.01	0.02	-0.63	-1.8	-0.04	1.56	1.28	-2.28	36.05
WPR	1336820.0	-50.58	20.87	-106.67	-77.23	-50.0	-24.14	14.29	0.03	-1.2
VOL1	1336820.0	2.23	2.75	0.0	1.47	2.3	3.68	21.74	13.35	235.14
VOL2	1336820.0	1.99	3.08	0.0	0.82	1.23	2.83	32.82	4.99	23.96
Delta_MA5	1336820.0	-0.04	0.38	-4.98	-9.88	-0.01	0.8	4.98	801.6	2.02

Table 1 - TITLON data set summary stat

There are some interesting insights into the summary statistics. The standard deviation of 'RSI6' and 'RSI12' are extremely large which indicates significant deviation from the mean of RSI values. The ‘Close’ variable also has a high value of 176.03 for standard deviation which indicates substantial price fluctuations, which can be normal due to high volatility of the stock market. Features like 'Close' and 'VOL1' show high variability with maximum values reaching 4900.00 and 21.74, respectively, compared to much lower minimum values, suggesting outliers or rare high-value instances. There are also extreme skewness and kurtosis in several features, particularly 'Close', 'BIAS5', 'BIAS10', and 'VOL1', suggest that the data contains significant outliers or anomalous values. After further analysis, we saw that these values stem from the extreme market events and cannot be discarded as an error.

2.2 Exploratory analysis

We need to explore our variables especially the response variable to gain valuable insights from the data. To start, we need to check the distribution of our response variable, the close price.

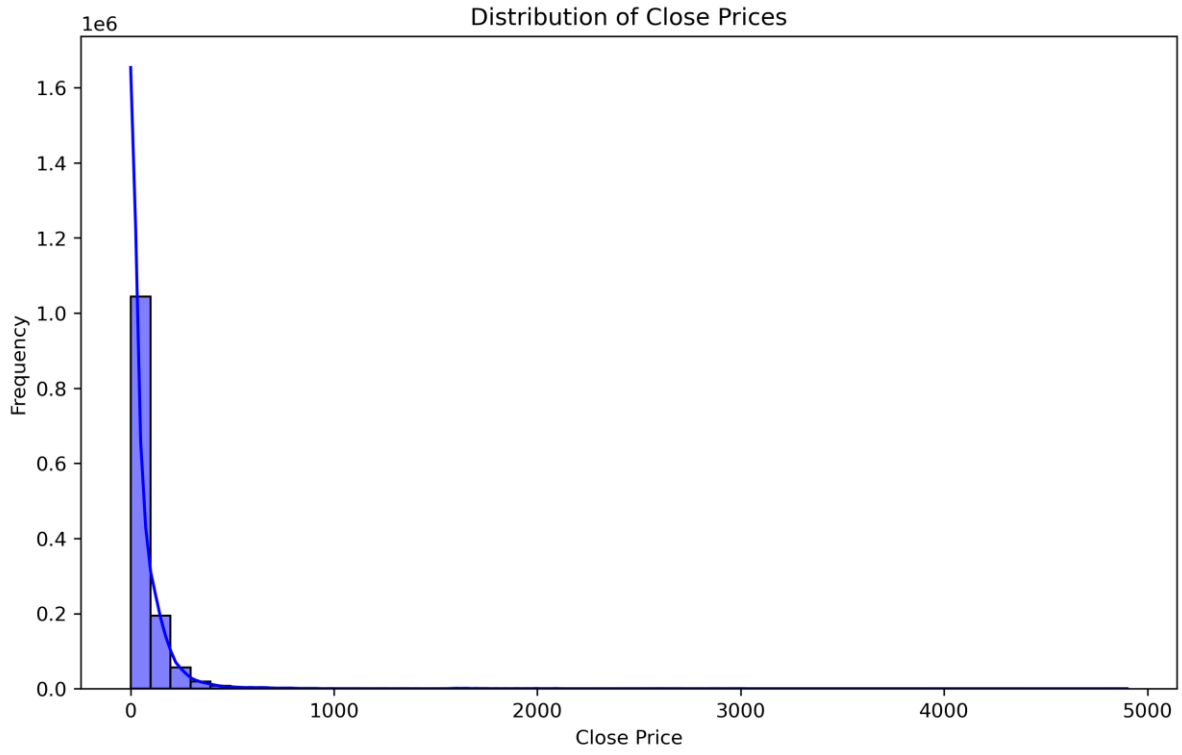

Figure 1- Close price distribution

In figure 1, you can see that the distribution is highly right skewed. This indicates that there are few stocks with extremely high prices compared to most stocks with relatively low prices. This makes the cluster at the beginning of the x axis with a long tail towards the end. It also shows that there are some outliers in the data that vary significantly compared to most of the prices. This phenomenon is normal in stock price data because stock market can be affected by numerous factors that make huge spikes in prices.

For checking outliers, one of the most helpful tools are boxplots as they can effectively show the variation range, percentiles and deviations from the median. Figure 2 shows the boxplots for all numerical variables in the dataset. The first one is the close price about which we talked earlier, this confirms figure 1 conclusion that most of the prices are clustered towards low values and some prices are extremely high. Fama French factors SMB, HML and LIQ are close to normal distribution with most of the values close to the median and some few outliers. The MOM (momentum) is also normal with a much smaller deviation from the median and most of the values are clustered around zero. MA5, MA10, MA20 (Moving Averages) show very similar behavior to the price with most of the values clustered towards the lower end and some outliers and this behavior is normal because they are calculated based on the price, so they inherit price patterns. BIAS5, BIAS10 (bias indicators) have a median of zero but they are significantly dispersed compared to other variables with lots of outliers that are the result of extreme market events. RSI6, RSI12 (Relative Strength Index) are mostly between 0 and 100, with some values close to boundaries that indicates overbought and oversold states and some outliers. Stoch_K, Stoch_D (Stochastic Oscillator) are also mostly between 0 and 100 and like RSI, reflects oversold and overbought states. The MACD and Signal are displaying a very wide range with lots of outliers, it indicates that there is a significant difference between short-term and long-term exponential moving averages. The WPR (Williams %R) ranges mostly from -100 to 0. The distribution looks to be uniform without outliers. VOL1, VOL2 (Volume Indicators) displaying a very wide range with lots of extreme values, indicating high trading volumes for some stocks in the dataset. And lastly the Delta_MA5 has a very wide range showing that 5-day moving average changes significantly which can be an indication of high volatility in the market.

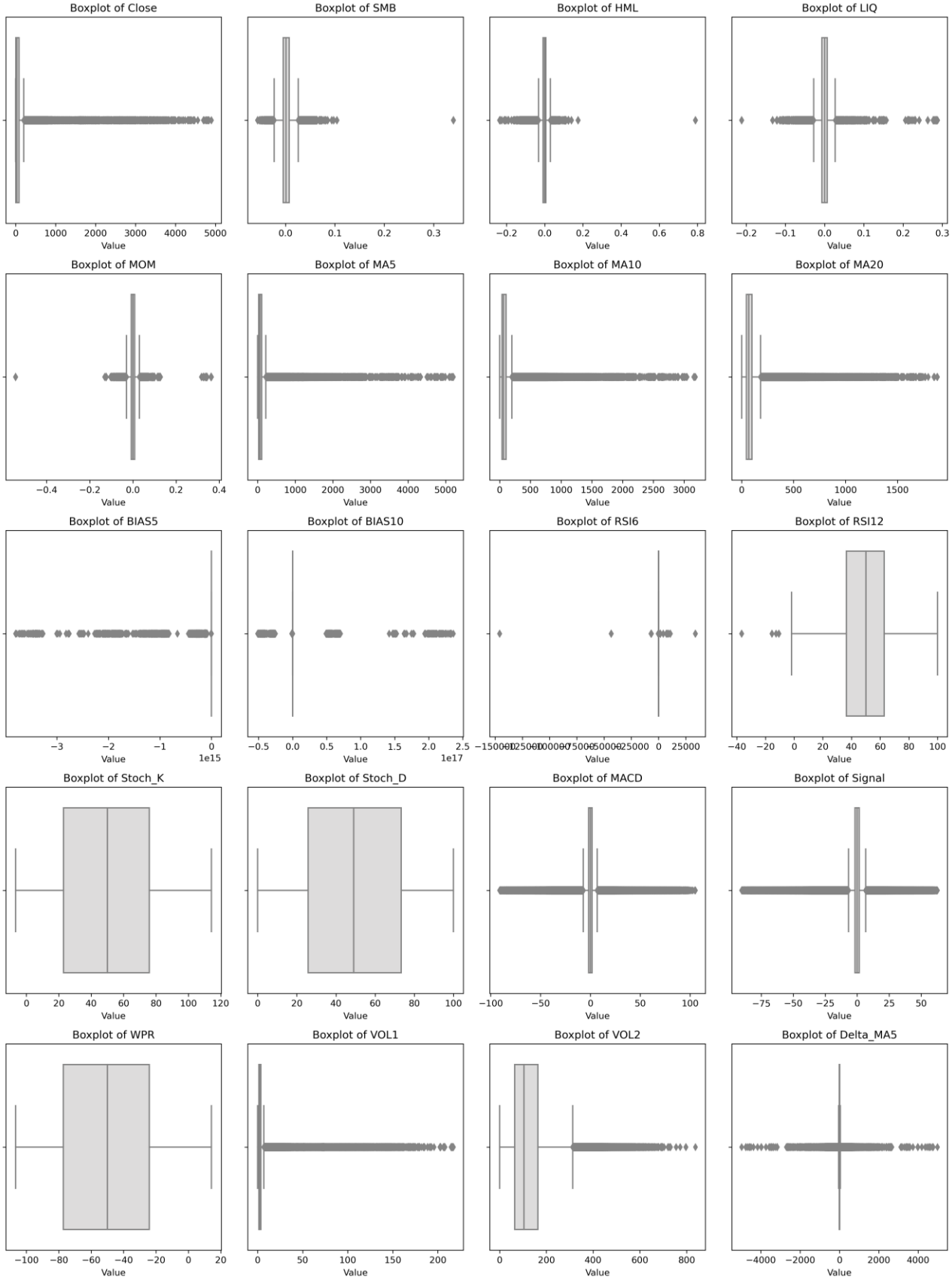

Figure 2- Boxplots of numerical varia

The first step of every machine learning or deep learning analysis is finding out the relationship between variables in the dataset as it is one of the most crucial factors that determines the model`s success or failure. One of the most powerful tools for examining these relationships is the correlation plot. The correlation plot, also known as heatmap, is a plot that shows correlations between variables in a matrix. The value 1 indicates high positive correlation and -1 indicates high negative correlation. Figure 3 shows the heatmap of our dataset.

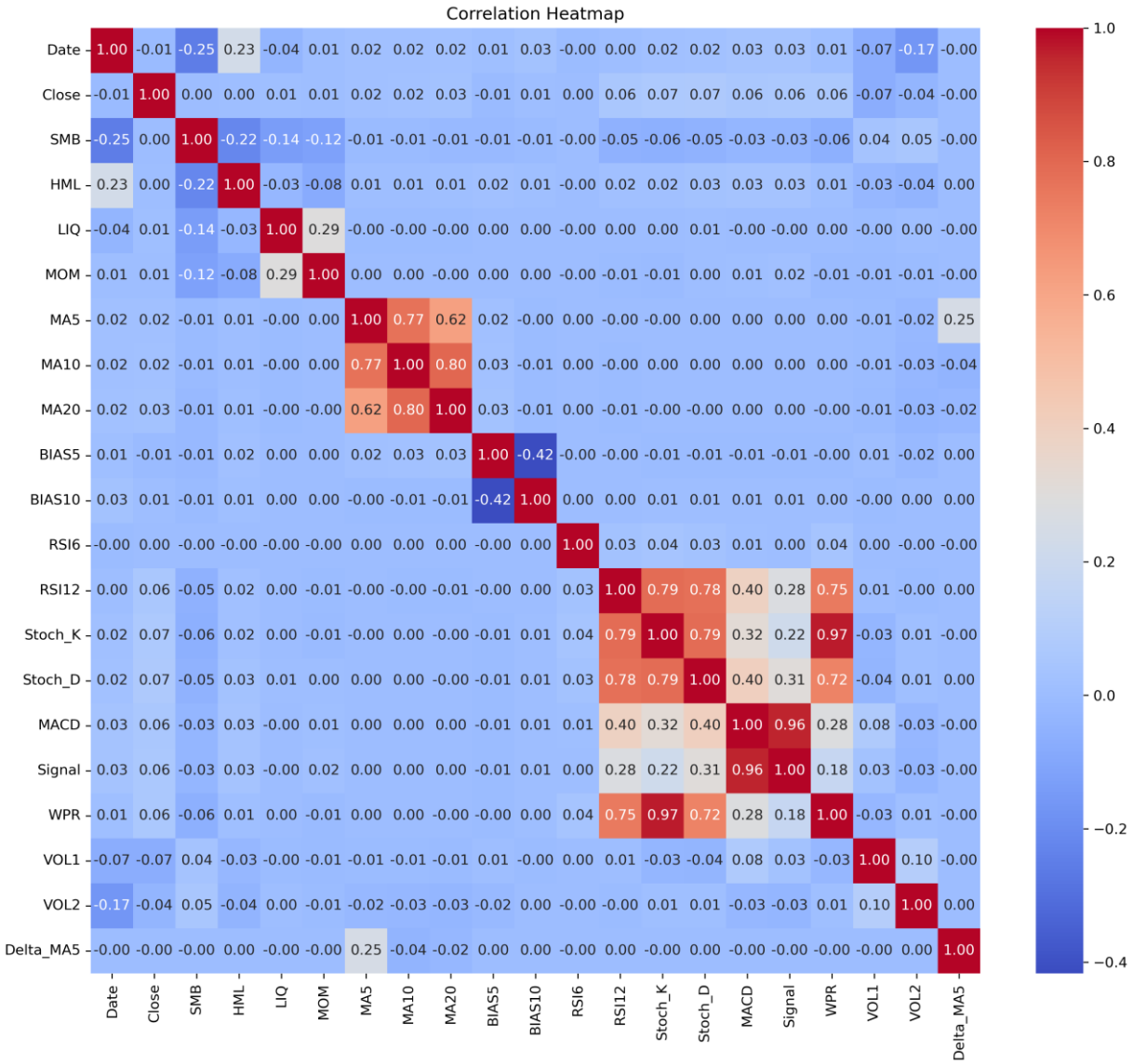

Figure 3 - correlation plot for all numerical variables in the dataset.

In figure 3, the closing price does not correlate significantly with any other variable, which is expected because the stock market is a very complex, high dimensional system that cannot be explained by a single variable. There are strong positive correlations among different moving averages and different Stochastic Oscillators that are also normal because they are calculated from the same base. The MACD and its Signal line have a strong correlation of 0.96, indicating that most of the time short-term and long-term exponential moving averages change together and do not deviate a lot. WPR (Williams %R) have an extremely high correlation with Stoch_K and Stoch_D, indicating that the oversold and overbought states are detected in the same way by these indicators.

2.3 Train-test split

In this study we decided to use the data from 1997-06-02 to 2021-01-01 as the training set and the data from 2021-01-01 to 2024-03-22 as the test set. The reason that we choose this method instead of random sampling method is that in the time series data random sampling can lead to data leakage because the model should not be given any information from the future that it is going to predict (Hyndman, R. J., & Athanasopoulos, G. 2018). Therefore, for the unbiased model evaluation we need to split data based on time. This split puts 1 percent of the data in the test set and the rest of the data in the training set. The data is scaled using the min-max scaler, making the features to be in a fixed range of 0 and 1.

3. Models

3.1 FT-Transformer: Detailed Explanation

FT-Transformer is the new method of operation on data in the tabular domain. It uses Transformer architecture but is reformed to work with structured data. It is, to be specific, a reforming model of the Transformer model, normally used in tasks to deal with natural language processing; this model is much like another model, AutoInt. FT-Transformer primarily focuses on transforming both the categorical and numerical data into tokens, which can be more easily processed by a stack of Transformer layers. The model design of FT-Transformer lies in the attention mechanism, similar to that applied in the AutoInt model for structured data analysis. It has the capability to transform the two forms of data: categorical and numerical into tokens, but which the AutoInt mainly focuses on the categorical data (Nguyen et al., 2023).

T-Transformer is a two-model model that contains two main components: the Feature Tokenizer and the Transformer. The Feature Tokenizer module transforms all the features—categorical and numerical—into tokens. This model uses an embedding layer for transforming categorical data into tokens and a dense layer for the numerical data. After that, the resultant tokens are fed to the Transformer component. The Transformer component of FT-Transformer is the same used in natural language processing for all other tasks. (Cho et al., 2022).

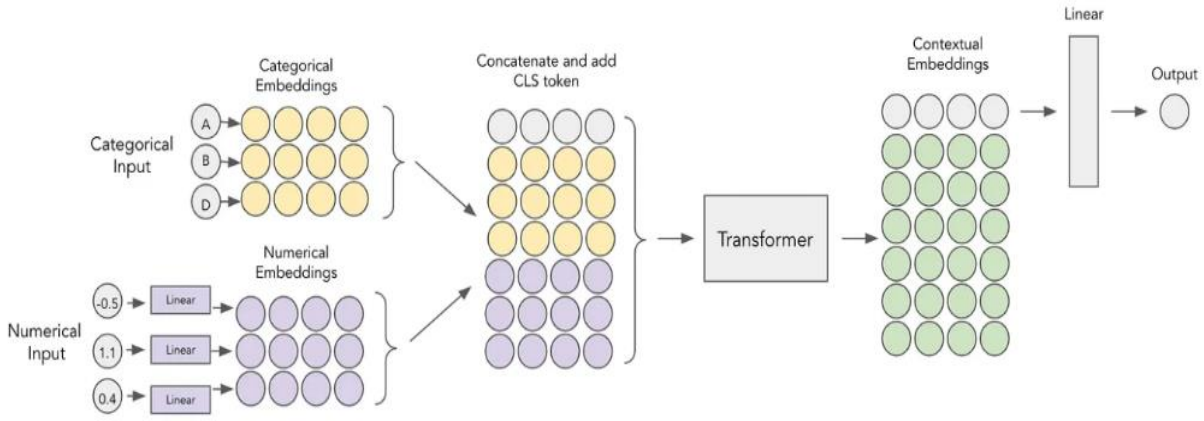

Figure 4- overview of FT-Transformer architecture (ResearchGate, 2023)

in the Transformer Layers The [CLS] token, used for classification tasks, is appended to the embeddings matrix T. The resulting matrix is then processed through multiple Transformer layers. Each Transformer layer consists of a multi-head self-attention mechanism and feed-forward neural networks, interspersed with layer normalization and residual connections.

FT-Transformer, by its nature, is powerful in the modeling of complicated cross-interactions because it captures the feature set and is innate to the architecture of self-attention (Gupta et al., 2023).

3.2 ResNet

In the context of CNNs, there is an underlying notion that 'the deeper, the better'—deeper models are theoretically more capable due to the increased parameter space. Once this process is repeated a couple of times, the degradation is deep, and performance drops drastically. This was a failure that was noticed in VGG networks. If we bypass the input to the output of the first layer, then the network should learn functions with the input added to them. This was designed to help tackle the gradient vanishing issue in very deep networks, where gradients become too small and thus do not update the weights effectively. ResNets bypass this by allowing gradients to flow directly through skip connections (He et al., 2015b).

It is known that as the number of layers in a network increases, gradients from backpropagation of the loss function with respect to the weights can get extremely small,

making it difficult for the latter layers to learn meaningful patterns. On the other hand, gradients can get very large, causing the training process to be unstable. The degradation problem: DNN performances saturated then degrading on the training data while network depth increased. This problem is highly counterintuitive because one would expect that deeper networks should have the capacity to extract more intricate and abstract features (Szegedy et al., 2016).

The core design and operation of ResNet are grounded in some significantly important building blocks, which address the problems of extremely deep neural networks. These core building blocks include residual blocks, skip connections, stacked layers, and global average pooling (GAP) (Szegedy et al., 2016). The basic unit among these blocks is the residual block (Szegedy et al., 2016):

H(x)=F(x)+x where F(x) denotes the residual mapping to be learned by a few stacked layers. The x term introduces identity mappings and, therefore, without any effort, the gradient flows smoothly. The research introduces the problem of degradation: in certain cases, as the network grows more profound, accuracy can deteriorate instead of improving. ResNets introduce shortcut connections that skip one or more layers. In this way, the network learns residual functions F(x)=H(x)+x instead of direct mappings H(x). It makes optimization easier and helps in training deeper networks. The +x operation is implemented by using a 'skip connection', which simply performs identity mapping, thus connecting the input of the subnetwork with its output. For these in recent work, the termed residual connection is named. The function F(x) typically is realized as some form of matrix multiplication intermixed with a few activation functions and some normalization operations (like batch or layer normalization). Each of these sub-networks is collectively called a 'residual block.' Just stacking these blocks in succession creates a deep residual network.

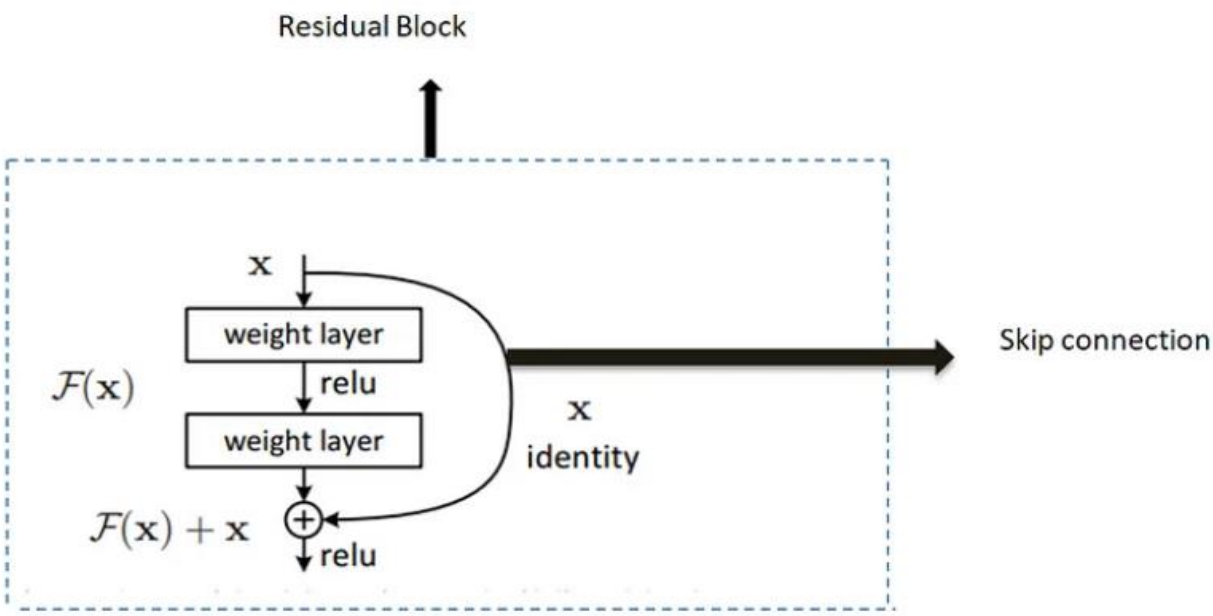

Figure 5 - How ResNet learns (He et al., 2015b)

Skip connections facilitate the creation of residual blocks. They simply bypass the input of the residual block through the convolutional layer and add it to the output of that residual block (Szegedy et al., 2015).

ResNet-34 is a network among the large family of Residual Networks (ResNet) models proposed in 2015 by Microsoft Research scientists. The core of ResNet-34 consists of several residual blocks, each holding two convolutional layers with 3 x 3 kernels. Batch normalization and ReLU activation follow every convolution operation (Szegedy et al., 2015).

Broadly, the residual blocks in the design of ResNet-34 give an avenue through which deeper neural networks can be constructed while averting the common challenges facing architectures at such depths. Since then, it has been a major hallmark allowing progress in deep learning, particularly in tasks requiring very deep networks (Simonyan and Zisserman, 2014).

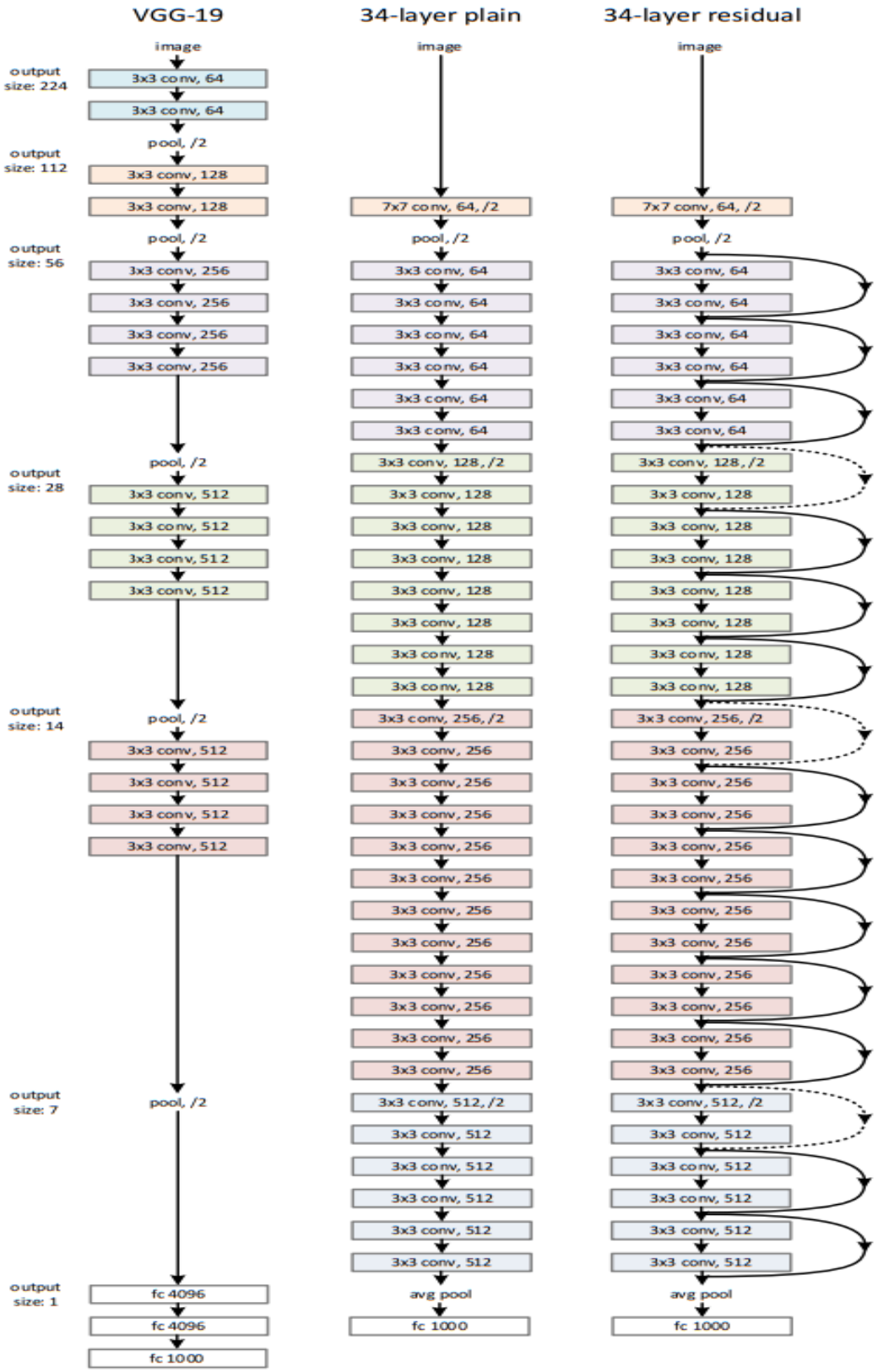

Figure 6 - ResNet-34 architecture (Sutskever et al., 2013)

To sum up, there are three advantages to ResNet: first, training very deep networks; second, faster convergence; and finally, reduced model complexity. The skip connections in ResNet make the training easier because the network can learn identity mappings, such that whenever it is needed, learning of the identity function becomes easier (Sutskever et al., 2013).

3.3 Multilayer Perceptron

The perceptron is a single-layer neural network that was originally designed to classify images, and it behaves as if its output were binary (true or false; represent real numbers). It takes weighted input and an activation function, then uses Stochastic Gradient Descent to optimize weights. However, it also suffers from limitations like an inability to solve non-linear problems such as the XOR gate. The Multilayer Perceptron (MLP) is used to overcome the limitations of the Perceptron. MLPs include the input, hidden, and output layers, which help to model non-linear relationships. Backpropagation is a new algorithm for weight adjustment, expressing error as the amount of change in weights that gradually approaches an actual minimum (Tolstikhin et al. 2021).

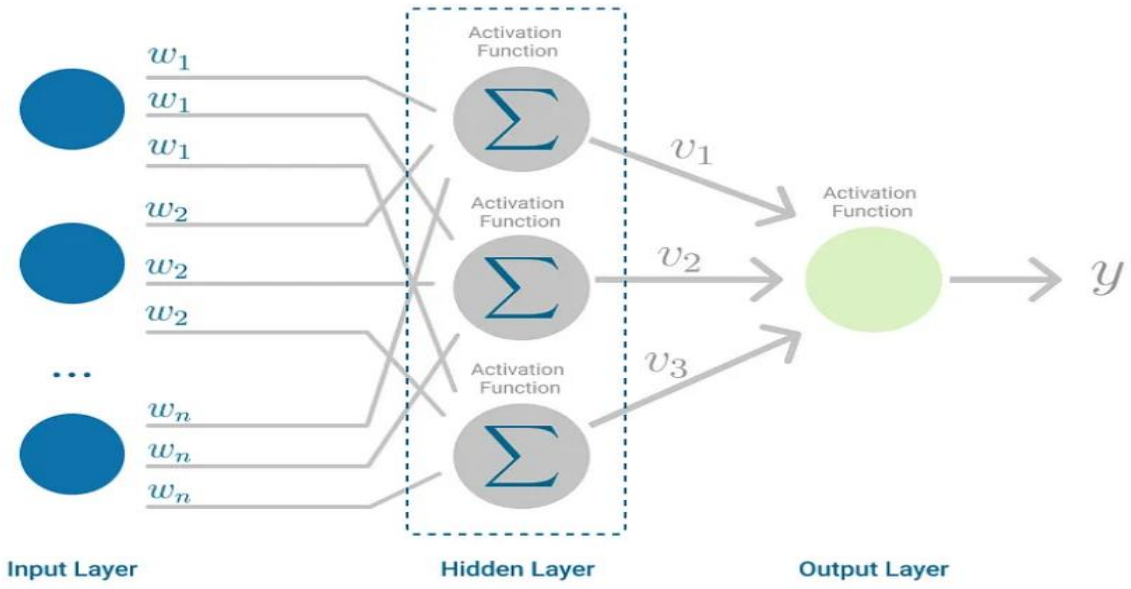

Figure 7- Overview of a feedforward neural network architecture showing input, hidden, and output layers (Hastie, Tibshirani, and Friedman, 2009).

Feedforward neural networks are called MLPs. This is called feedforward because the inputs are fed forward and transformed into an output after applying weight, sum, and activation. A feedforward neural network (FNN) is an artificial neural network whereby connections between the nodes do not form a cycle. The information can only flow in one direction (Akiba et al., 2019).

Multilayer Perceptron (MLP): This is an advanced neural network capable of solving non-linear problems unlike single-layer Perceptrons. These different layers of neurons include an input layer, one or more hidden layers, and finally the output layer. When doing a feedforward pass, each neuron processes its inputs similar as we did it before: first performing the weighted sum and then passing through an activation function (Bello et al., 2021). In this way, the network can make predictions based on input data from the layer at which to start executing.

Activation functions are a key feature of MLPs because they introduce non-linear behavior to the network, which allows it to model complicated patterns. Activation functions such as the Rectified Linear Unit (ReLU) and sigmoid functions are provided (Tolstikhin et al., 2021).

This learning mechanism is called backpropagation, and it simply corresponds to the fact that MLPs are trained in an iterative way where the network weights are adjusted so as to minimize a cost function (Bello et al., 2021).

Backpropagation helps us to do the optimization process, and Gradient Descent is its corresponding algorithm that minimizes our cost function. This is done by incrementally changing the weights within the network. The algorithm initializes weights with random initialization, computes the gradient of the loss function for each training example, and updates weights by subtracting a learning rate. The weights are updated in small steps to ensure that the network slowly converges to minimum error.

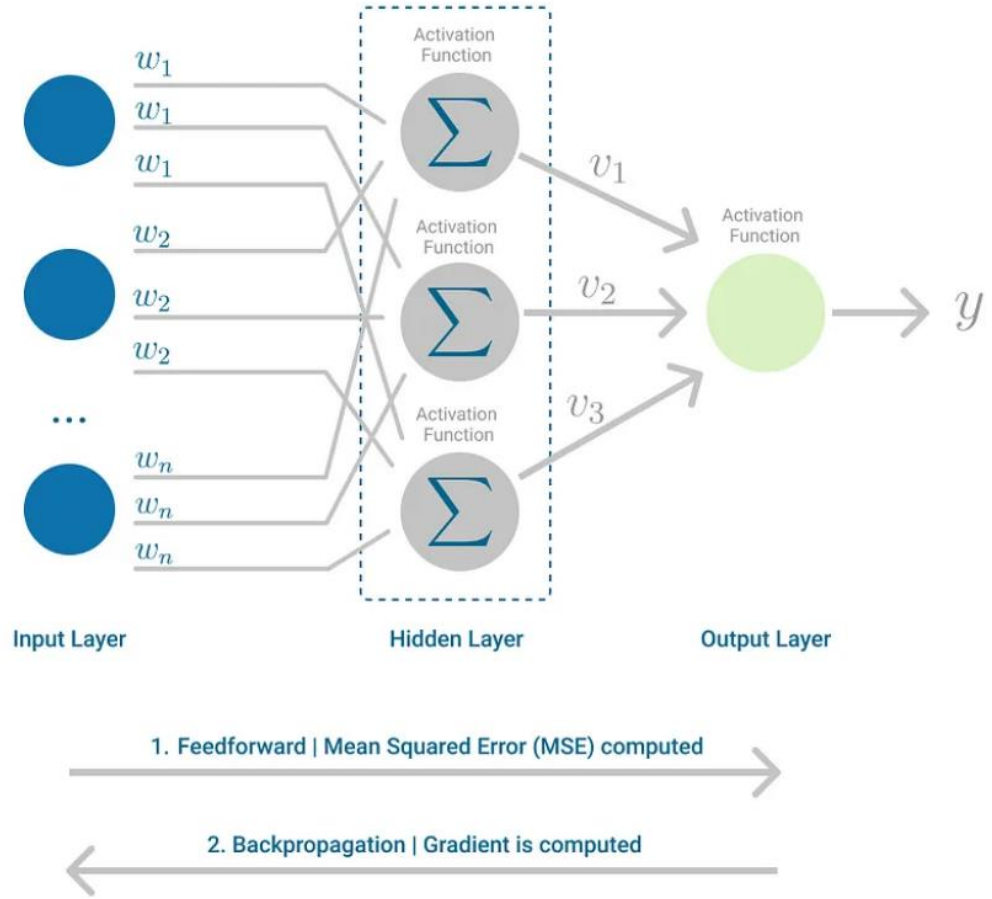

Figure 8 - Multilayer Perceptron, highlighting the Feedfoward and Backpropagation steps (Gorishniy, Rubachev, Khrulkov, & Babenko, 2021).

But MLPs have their own problems, too. Despite its high improvement in accuracy over traditional classification methods, they suffer from the prohibitive computational cost of training these networks significantly increasing when deploying large and complex datasets. We experience growth costs and may need significant computational resources for such a task. In addition to this, MLPs have a key downside—they require perfect training, otherwise the model will completely fail. While it seems simplistic to some, if the model is not trained well, this can lead to a variety of performance issues that decrease confidence in our prediction accuracy and reliability. And also the other problem that we have with Multilayer Perceptrons is Dense Connectivity where again results in heavy parameters and Redundant Nodes as above. It becomes hard to manage and optimize the model, which results in inefficient working of it. Nevertheless, exactly these challenges can often be overcome by the benefits of MLPs on suitable problems and datasets (Bello et al., 2021).

3.4 XGBoost

In fact, XGBoost is short for "Extreme Gradient Boosting," referring to this paper: *Greedy Function Approximation: A Gradient Boosting Machine* by Friedman. Gradient Boosting fits into this family, and the approach of picking it up with Gradient boosted trees is well known since quite some time, however differs from the above traditional techniques to explain GBM. A more concise, proper explanation which also gives a hint about the model formulation that XGBoost (Chen and Guestrin, 2016) uses to solve the optimization.

GBDT is another ensemble learning algorithm that creates multiple decision trees one after the other, with each new tree correcting errors done by its previous ones. The gradient boosting focuses more on the errors from previous iterations, making it less biased and capable of reducing underfitting (Chen and Guestrin, 2016).

XGBoost is a type of supervised learning method used for solving regression, classification, and ranking problems. The basic concept of supervised learning is reviewed before moving on to (algorithm) trees in particular (Chen and Guestrin, 2016).

In supervised learning, the task of training a model involves finding the best parameters θ that fit the training data x_i and labels y_i. To achieve this, we need to define an objective function that measures how well the model fits the training data.

A salient characteristic of objective functions is that they consist of two parts: training loss and regularization term.

The objective function is expressed as:

$$obj(\theta)=L(\theta)+\Omega(\theta) \qquad (5.1)$$

where L is the training loss function, and Ω is the regularization term. The training loss measures how predictive our model is with respect to the training data. A common choice for L is the mean squared error (MSE) and Another commonly used loss function is logistic loss, used for logistic regression.

The regularization term, $\Omega(\theta)$, controls the complexity of the model, helping to avoid overfitting. It ensures the model remains generalizable by preventing it from becoming too complex (Chen and Guestrin, 2016).

Ensembles of Decision Trees — XGBoost: Notable is the fact that we use gradient-boosted ensembles and individual classification/regression tree (CART) models to achieve our final analysis model. A tree ensemble model is a group of classification and regression trees (CART). The output of all the ensemble trees is summed up to give a final prediction (Chen et al., 2015).

One tree is too weak for any usable purpose. A collection of trees is used instead to get potential outcomes and then sum them up. For example, in an ensemble consisting of 2 trees, the prediction scores of each tree are summed together for a final score. These two trees are complementary to each other, and together they contribute more power in making better predictions for the model (Chen et al., 2015).

For all supervised learning models, the process involves defining an objective function and optimizing it. This objective function always includes training loss and regularization, balancing prediction accuracy with model complexity.

The parameters are the functions f_i, each containing the structure of the tree and the leaf scores. Learning tree structure is more complex than traditional optimization problems because it is impractical to learn all trees at once. Instead, an additive strategy is used: fix what has been learned and add one new tree at a time. The prediction value at step t is written as $y_i^{(t)}$. At each step, the goal is to add a tree that optimizes the objective function.

Complexity: Tree complexity is defined to constrain overfitting. It is more lightweight, so it reduces the tree to a definition that relies on a leave score vector as you call and what function maps data points into leaves and also how many maximum numbers of leaf nodes. This exact definition facilitates an intuitive understanding along with the optimization of the model.

Structure score: It indicates the adequacy of tree structure. It gathers statistics at the leaves, sends them up on demand, then sums those stats and computes scores via a hard-coded formula (Chen et al., 2015). To optimize a level of the tree, one—leaves is divided into two to calculate gain. An optimal split can be found by searching over the sorted instances from left to right for a good or best point.

In the training step, a crucial component is the regularization term, which helps control the complexity of the model. To define the complexity of a tree f(x), we first refine the definition of the tree. the objective becomes:

$$\text{obj}^{(t)} = \sum_{j=1}^{T} \left[G_j w_j + \frac{1}{2}(H_j + \lambda) w_j^2 \right] + \gamma T \quad (5.12)$$

In this equation, w_j are independent with respect to each other. The form $G_j w_j + \frac{1}{2}(H_j + \lambda) w_j^2$ is quadratic, and the best w_j for a given structure q(x) and the best objective reduction we can get is:

$$w_j^* = -\frac{G_j}{H_j + \lambda}$$

$$\text{obj}^* = -\frac{1}{2} \sum_{j=1}^{T} \frac{G_j^2}{H_j + \lambda} + \gamma T \quad (5.13)$$

This last equation measures how good a tree structure q(x) is.

XGBoost 1.0.0: Support for Saving and Loading Models in JSON. Other than being expressive to human users, through means of simple key-value pairs such as {…} type serializations or non-human-readable unstable binary format (DMLC provides also implementation) XGBoost decided to introduce its native support for parsing model architecture in NImeynesson mearKey that looks like this:- parlolouses — a smaller dictionary->ParameterValue that conveys meaning about lightweight... Version 1.6.0 later improved this format by using UBJSON for serialization efficiency, too.

Taking memory snapshots is also supported for cases such as distributed training. This saves the entire state of training, also makes fault tolerance easier, and with SCALED allows you to re-train in a short period. But the memory snapshot format itself is neither backward compatible nor meant for long-term storage, and one should use it as checkpointing rather than state management. Since user-defined objective and metric functions are language-specific, we do not include them in the model file (He et al., 2014).

XGBoost provides an efficient and scalable implementation of the Gradient Boosting algorithm. XGBoost scales well to large datasets and can be easily distributed across a

cluster of machines, enabling it to integrate within big data processing pipelines like Apache Spark. XGBoost models process on GPUs with minimal host that allows you to train trees as low as one minute utilizing all existing cores in just a single computer (He et al., 2014). XGBoost is an optimized distributed gradient boosting system designed for natural language processing, metrics prediction, and other machine learning tasks.

3.5 CatBoost

3.5.1 Introduction

CatBoost is a Yandex-developed gradient boosting algorithm that can treat categorical features as first-class citizens in an efficient way. CatBoost allows you to input unprocessed categorical features directly and can train models with high accuracy, saving time on making your data more structured. This algorithm uses techniques like ordered boosting to create stronger models and prevent the overfitting issue (Prokhorenkova et al., 2018).

Because CatBoost supports all native features, including numeric, categorical, and text, this saves a lot of time from doing traditional pre-processing on your data. In addition, CatBoost uses statistic-based encoding with target and random permutation to handle high cardinality features effectively (Prokhorenkova et al., 2018).

CatBoost—Text Facts: For multiclass classification, the library comes with its automated text cleaning and preprocessing built-in using approaches like Bag-of-Words (BoW), Naive-Bayes, BM-25, etc. These methods generate a dictionary (letters, words, grams, and so on) from text data, extracting words by indexes, and converting them into numeric features (Prokhorenkova et al., 2018).

Typically in the comparison of gradient boosting models, a few main algorithms are considered. Scikit-Learn provides an implementational standard called GradientBoostingClassifier that is used as a baseline for performance comparison. It is a trusted, scalable, and high-performance tree boosting library that can handle all the challenges in machine learning competitions as well as special use cases

	CatBoost	LightGBM	XGboost
Categorical Features	Automatc Categorical Feature handling. No need of preprocessing	Supports one-hot encoding, categorical features directly	Requires preprocessing
Tree Splitting Strategy	Symmetric	Leaf-wise	Depth-wise
Interpretability	Feature importances, SHAP	Feature importances, split value histograms	Feature importances, tree plots
Speed and Efficiency	Optimized for speed and memory	Efficient for large datasets	Scalable and fast

Figure 9 - comparison gradient boosting models

However, what is unusual about CatBoost is its support for quantile regression, not just classification tasks and vanilla data regression. Because quantile regression can predict possible values and give us an idea of how uncertain we are, it is especially helpful when dealing with a field like finance, where knowing the potential locus of future measurements may be important (Prokhorenkova et al., 2018).

CatBoost vs. Ferov & Modrý 2016: The feature importance calculated by CatBoost is based on SHAP (SHapley Additive exPlanations), which decomposes the prediction value into contributions from each of a set of features. It explains the model by measuring the impact of each feature regarding a single prediction value compared to baseline localization, which presents us with visual and very detailed explanations about highly positive/negative features affecting the outcome. You can apply SHAP on two levels: instance level or dataset-wide (Ferov & Modrý 2016).

Finally, the force plot represents different features and how they affect model prediction with respect to a specific instance. Check out CatBoost visualizations from SHAP, which in turn help to interpret and make sense of model decisions (Prokhorenkova et al., 2018).

Another great benefit is the fact that CatBoost provides advanced visualization tools. Visualization tools for the training process, model prediction, and feature importance are available to provide users with visual insights . CatBoost supports out-of-the-box fast scalable GPU training, which can quickly train models using an optimized-for-GPU gradient boosting algorithm, with the ability to work on multi-card configurations (Ferov & Modrý 2016).

One of the other strengths is integration with popular machine learning frameworks, such as Scikit-Learn, TensorFlow, and Keras. CatBoost is well known for its power to integrate into existing architectures in the machine learning field, like Scikit-Learn, TensorFlow, or deeper by implementing Conv2D layers using Keras API. CatBoost also has a peculiar symmetric weighted quantile sketch (SWQS) algorithm to handle missing values in an optimal manner (Prokhorenkova et al., 2018).

To sum up, there are additional benefits provided by CatBoost that should be considered, especially when it comes to categorization data.

3.6 Gradient Boosted Decision Trees

Jerome H. Friedman describes Gradient Boosting in the study titled “Greedy function approximation: a gradient boosting machine”. In his paper, Friedman describes the Gradient Boosting technique. Since it is a supervised technique, we begin with a set of input values x_i , and expected output values y_i, i=1,2,…. Gradient boosting takes the approach of iteratively constructing a collection of functions f_t, given a loss function L. Here we would like to emphasize that L has two input values, the $i-th$ expected output value yi, and the t-th

function f_t that estimates y_i. Assuming we have constructed function f_t we can improve our estimates of f_t by finding another function h_t such that $f_t + 1 = f_t + h_t$ minimizes the expected value of the loss function (Prokhorenkova et al., 2018). That is,

$$\widehat{h_t} = \arg\min_{h_t \in H} E\left[L\left(y, f_t(x) + h_t(x)\right)\right] \quad (6.1)$$

Where H is the set of candidate Decision Trees we are evaluating to choose one to add to the ensemble. Furthermore, by the definition of h_t, we can write the expected value of the loss function L in terms of y and f_t:

$$E\left[L\left(y, f_t(x) + h_t(x)\right)\right] = E\left[L\left(y, f_t(x)\right)\right] + E\left[\frac{\partial L(y, f_t(x))}{\partial f_t} h_t(x)\right] + \frac{1}{2} E\left[\frac{\partial^2 L(y, f_t(x))}{\partial f_t^2} h_t(x)^2\right] \quad (6.2)$$

One may notice that the right-hand side of Eq. (2) implies we wish to minimize the loss function's value on y and f_t plus something. If we assume L is continuous, and differentiable, we can add something related to the rate of change of L to f_t to shift its value somewhere in the direction that L is decreasing.

Under these assumptions then we can write a reasonable approximation for h_t (Prokhorenkova et al., 2018) :

$$\widehat{h_t} \approx -E\left[\frac{\partial L(y, f_t(x))}{\partial f_t}\right] \quad (6.3)$$

We refer to this technique as Gradient Boosting because we use the partial derivatives (gradients) of the loss function L with respect to the function f_t to find h_t. Prokhorenkova et al. point out that we may not have an easy way to compute $E\left[\frac{\partial L(y, f_t(x))}{\partial f_t}\right]$. This could be because it would be difficult, in general, to say what the probability of specific values of x

should be, and we may not know what f_t should be because we could be using stochastic techniques, such as some algorithm to construct a Decision Tree to define f_t. However, we can assume, as Prokhorenkova et al. suggest,

$$E\left[\frac{\partial L(y,f_t(x))}{\partial f_t}\right] \approx \frac{1}{N}\sum_{i=1}^{N}\frac{\partial L(y_i,f_t(x_i))}{\partial f_t} \quad (6.4)$$

Although we are covering Friedman's Gradient Boosting Decision Trees technique in this section, we use this reference to Prokhorenkova et al. in our explanation, since our ultimate goal is to provide the reader a clear understanding of CatBoost (Prokhorenkova et al., 2018).

For GBDT's the base case f_0 is a Decision Tree, and the f_t are also Decision Trees. When we add a Decision Tree to construct f_{t+1} in this manner, the expected value of the loss function L shrinks, implying that the estimates f_{t+1} are better than the estimates f_t (Prokhorenkova et al., 2018).

3.6.1 CatBoost Gradient Boosted Trees Implementation

Prokhorenkova et al. suggest CatBoost and compare it with XGBoost and LightGBM. Their write-up on the CatBoost learner explains how they have optimized the GBDT algorithm Friedman discussed.

The second alteration made to Gradient Boosting via CatBoost is how it handles high cardinality categorical items. CatBoost uses one-hot encoding for low cardinality categorical variables. This definition of low cardinality is somewhat loose and can vary depending on the type of computing environment being used or whether you use CatBoost in a specialized mode.

Prokhorenkova et al. use the term "Ordered Target Statistic" to refer to the technique CatBoost uses for encoding categorical variables, when CatBoost is not using one-hot encoding. Micci-Barreca introduces target statistics in "A preprocessing scheme for high-cardinality categorical attributes in classification and prediction problems". CatBoost encodes categorical values in order to alleviate the problem of target leakage. Prokhorenkova

et al. write that CatBoost avoids target leakage because the technique it uses for encoding categorical variables has a certain property. (Prokhorenkova et al., 2018).

The second property of the Ordered TS that Prokhorenkova et al. describe is that it will eventually consume all training examples S. With this property, we can encode our categorical values in a way that contains all information in the data as soon as possible after N iterations. This balances the overfitting protection of the first property, ensuring we are not underfitting, while also implying convergence because we can effectively use any or all available data.

Oblivious Decision Trees (ODTs) are also important to understand for CatBoost and relating them with some understanding of core concepts taught in ML. Under the hood, CatBoost constructs an ensemble of ODTs. ODT (Overlap Dispersion Tree) is a full binary tree, so if the number of levels of ODT is n, it has 2^n nodes. Another unique thing is that all non-leaf nodes of the ODT have an identical split criterion. ODTs "…are balanced, less prone to overfitting, and allow speeding up execution at testing time…" according to Prokhorenkova et al.

This sensitivity to the hyper-parameter setting is something that researchers should be mindful of as a way in which CatBoost experiments can differ from traditional gradient boosting. Perhaps discrepancies in running time complexity are related to improperly chosen values of this hyper-parameter (Prokhorenkova et al., 2018).

3.7 Self-Normalizing Neural Networks (SNNs)

One of the great recent developments in deep learning, especially for feed-forward neural networks (FNNs), was Self-Normalizing Neural Networks (SNNs). Despite the dramatic success of convolutional neural networks (CNNs) in computer vision and recurrent neural networks (RNNs) for natural language processing, fully connected feed-forward networks have struggled to outperform previous techniques on these challenging datasets.This shortcoming was addressed by SNNs using an automatic normalization mechanism that keeps neuron activations stable and allows the training of deep networks (Klambauer et al. 2017).

SNNs have at their foundation the scaled exponential linear unit (SELU) activation function, which induces a self-normalizing effect where the output of neurons is pushed towards a zero mean and unit variance. This property eliminates the need for explicit normalization techniques like batch norm, which is a large operation and can be perturbed by changes in stochastic gradient descent. It ensures that activation distributions are stable across the layers in SNNs, which is critical to effectively training deep networks.

Despite this simplicity, several benchmarks have shown that networks of these neurons yield superior performance relative to conventional rate encoders. In addition to plain vs sybil, we investigated SNNs against different normalization methods for FNNs such as batch norm, layer norm, weight norm, and even highway/residual networks.

One of the strong sides of SNNs is that they could be effectively trained with very deep networks. Batch normalization can bring high variance in training errors to FNNs, often leading to unsuccessful outcomes (Klambauer et al., 2017).

Theorem 1 proves that under specific parameter settings and for certain intervals of the inputs, mean and variance activations converge to stable fixed points, avoiding exploding or vanishing gradients. Theorem 2 states that the variance ν of neuron activations in SNNs is bounded from above. This means that when the variance is mapped across many layers, it will be kept within an upper limit, preventing exploding gradients and ensuring that learning remains stable (Klambauer et al., 2017).

Theorem 3 complements this by ensuring that the variance ν is also bounded from below. This lower bound prevents the variance from becoming too small, thereby avoiding vanishing gradients. these three theorems collectively ensure that SNNs maintain stable mean and variance of neuron activations, which is crucial for robust and effective training of deep neural networks.

The number of neurons will be much too large for the output to approximate a normal distribution (assuming CLT applies); hence, this assumption justifies deriving the mapping g function and self-normalizing properties of SNNs (Bengio, 2013).

Finally, SGD, Momentum, and Adamax are the most popular optimizers for training SNNs (as demonstrated empirically by Klambauer et al., 2017), followed very closely by Adadelta;

we observed that adapting using an Adam optimizer requires parameter tuning to work properly.

3.8 GrowNet

A new gradient boosting framework, GrowNet, utilizes shallow neural networks as weak learners, rather than the traditional decision trees. This allows GrowNet to incorporate general loss functions applicable across various tasks such as classification, regression, and learning to rank. A key innovation is the inclusion of a fully corrective step, which mitigates the shortcomings of greedy function approximation found in conventional gradient boosting decision trees (GBDTs). The model has shown superior performance compared to state-of-the-art boosting methods on various datasets. An ablation study has also highlighted the importance of various model components and hyperparameters (Badirli et al., 2020).

GrowNet aims to address these challenges by introducing a different paradigm that constructs neural networks layer by layer using gradient boosting. Gradient boosting is known for building complex models by combining simpler models. Popular GBDT frameworks like XGBoost, LightGBM, and CatBoost use decision trees as weak learners to create robust models. However, decision trees may not always be the best choice, particularly for structured data, where neural networks often perform better (Badirli et al., 2020).

GrowNet combines the strengths of gradient boosting with the flexibility of neural networks, proposing a framework where shallow neural networks are used as weak learners (Bao et al., 2019).

The innovation includes a new optimization algorithm that converges faster and more efficiently than traditional deep neural networks. Major technical progress was made with the use of second-order statistics and global corrective steps, which stabilize models and improve target divergence. This off-the-shelf strategy simplifies training, making it both successful and feasible for different applications (Bao et al., 2019).

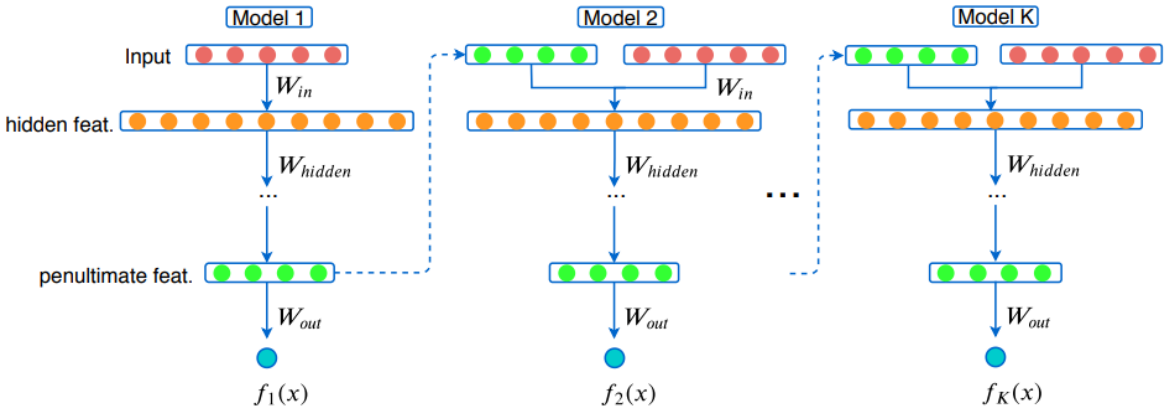

Figure 10 - GrowNet architecture (Badirli et al., 2020).

In GrowNet architecture, The process begins with the initial model trained on the input features. Subsequent models, or predictors, are trained on a combination of the original input features and the penultimate layer features from the previous weak learner. This stacking of features enables the network to incrementally leverage more complex representations. The final output of the model is calculated as a weighted sum of the outputs from all the individual predictors ($\sum_{k=1}^{k=K} \alpha_k f_k(x)$), ensuring that each new addition enhances the overall predictive performance. Here Model K means weak learner K (Badirli et al., 2020).

3.8.1 Model Description

GrowNet is a framework that combines gradient boosting with shallow neural networks as weak learners to create a stronger, higher-order model. In each boosting step, the model appends the original input features with the activations from the penultimate layer of the current iteration. This enhanced feature set then trains the next weak learner, focusing on the current residuals. The final output is a weighted combination of scores from all the sequentially trained models. Consider a dataset with n samples in d-dimensional feature space $D = \{(x_i, y_i)\}_{i=1}^{n}$, where $x_i \in R^d$ and $y_i \in R$. GrowNet predicts the output using K additive functions (Badirli et al., 2020) :

$$\hat{y_i} = E(x_i) = \sum_{k=0}^{K} \alpha_k f_k(x_i) \quad (8.1)$$

Here, $\mathcal{F}$, represents the space of multilayer perceptron, and $\alpha_k$is the step size or boost rate.

Each function f_k is an independent shallow neural network with a linear output layer. This streamlined approach, validated through empirical studies, demonstrates GrowNet's robustness and effectiveness in various machine learning tasks, offering a significant improvement over traditional method (Bao et al., 2019).

3.9 NODE

Neural Oblivious Decision Ensembles (NODE) is a novel deep learning technique for structured data, combining the advantages of both traditional methods like gradient-boosted decision trees (GBDTs) and deep neural networks (DNNs). While DNNs often struggle with structured data compared to GBDTs, NODE merges these strengths, offering a versatile regression model. NODE performs end-to-end, gradient-based optimization over generalized ensembles of Oblivious Decision Trees (ODTs), blending representation learning with the generative power of deep networks.

The NODE architecture extends a model based on oblivious decision trees (ODTs) where the same splitting feature and threshold is used at every depth of the tree. In particular, this approach ensures fast inference and helps to prevent overfitting due to the non-decreasing nature of structured functions (as opposed to vanilla neural networks), while the differentiation achieved by NODE enables it to be included in conventional deep learning frameworks such as PyTorch or TensorFlow. The architecture is intended to support multi-layered structures that allow more complex decision-making processes from deeper levels and enable the capturing of deep data dependencies (Popov et al., 2020).

NODE is a mostly differentiable model relying on the entmax transformation for making decisions in alternative deep learning with its close form Node Shifter solution. This conversion makes possible efficient sparsity and feature selection enabled by L1-L2 regularization, while retaining the simplicity of traditional decision trees (Huang et al., 2017).

Figure 11: An ODT within the NODE layer shares splitting features and thresholds across all internal nodes at a given depth. The tree outputs a sum of leaf responses weighted by an indicator function that defines the choice of feature split (Huang et al., 2017). Specifically, to

make the tree output differentiable, non-differentiable decision rules are replaced with continuous counterparts. This includes feature selection with \alpha-entmax transformation, enabling efficient learning of sparse choices using gradient descent.

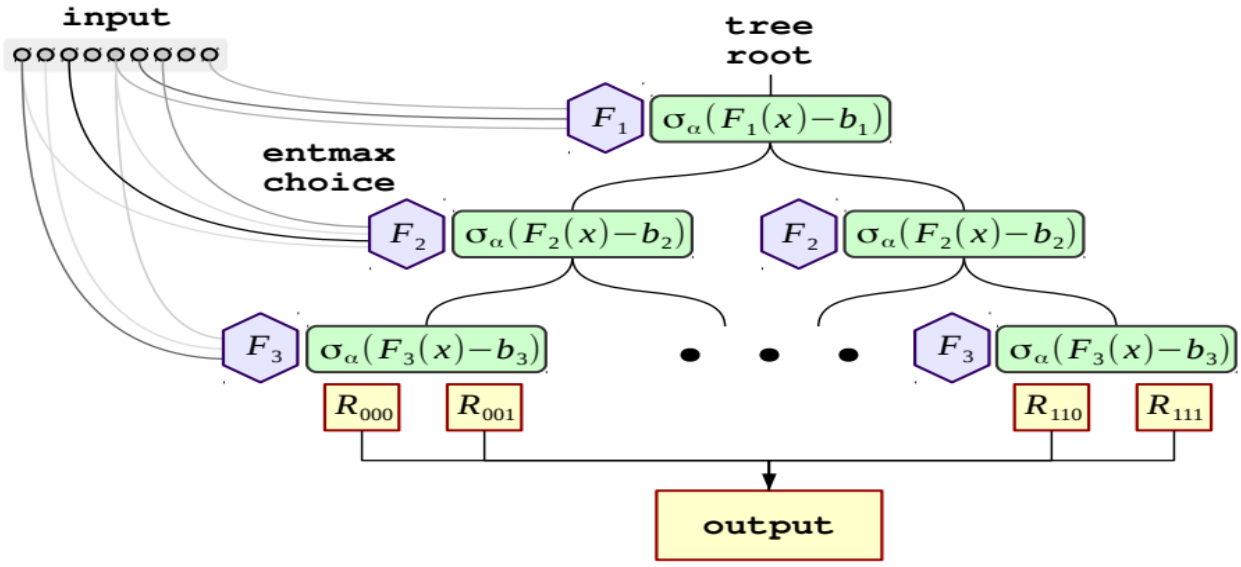

Figure 11- Single ODT in the NODE Layer (Popov, Morozov, & Babenko, 2019).

Figure 12 shows the NODE architecture's ability to create multi-layer structures, resembling DenseNet. Each layer in the NODE architecture is represented as a set of trees whose outputs are concatenated and passed to the next layers. The final prediction is the average of all tree outputs across all layers.

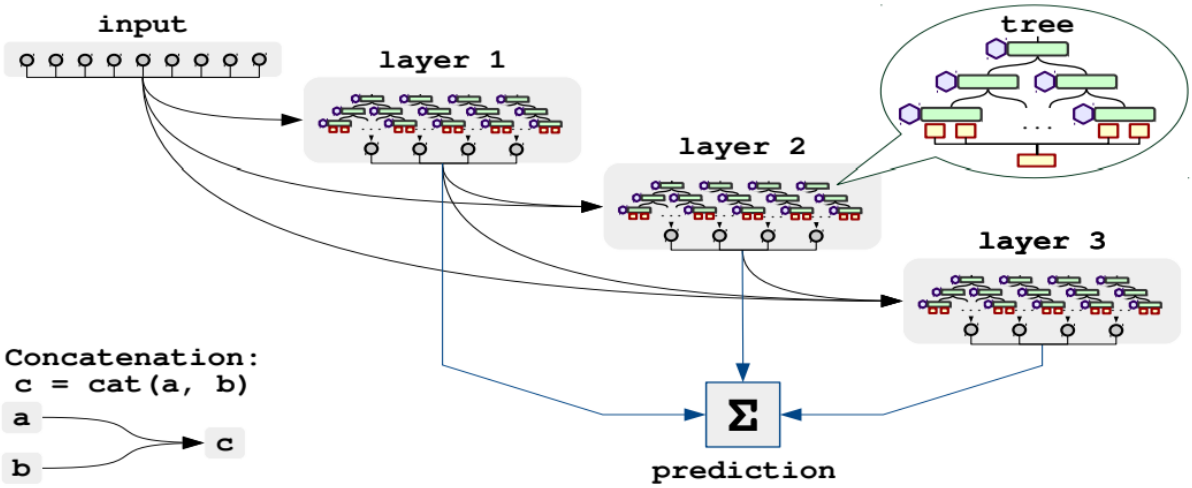

Figure 12 - Going Deeper with the NODE Architecture (Popov, Morozov, & Babenko, 2019).

The final architecture is flexible in terms of the number of output dimensions and can be useful for any type of learning (regression, classification… you name it!). This is controlled using the NODE model by averaging predictions across all layers, which helps avoid making an overconfident decision in any specific layer. This comes in especially handy when working with tabular data, where simple deep learning models usually fail to outperform gradient boosting methods (Huang et al., 2017).

3.10 DCN V2

The model we are going to explain is Deep & Cross Network (DCN), a structure specially designed for feature interaction learning, which is crucial in building successful recommender systems. However, in web-scale traffic scenarios with billions of training examples, DCN was observed to be less expressive. Recognizing this issue, DCN V2 was introduced to make it more usable for large-scale deployments associated with industrial use-cases.

DCN-V2 (Deep & Cross Network Version 2) is an advanced model architecture—a new generation of the deep and cross network that can be used to learn first-order, second-order feature interactions explicitly, as well as high-level representations. The specific components include an embedding layer and a cross network, illustrated with multiple layers when modeling explicit feature interactions. These cross layers are combined with a deep network that can learn implicit feature interactions. Taken together, this combination allows DCN-V2 to exploit complex but explicit cross terms in web-scale production data while preserving a neat formula for practical use (Wang et al., 2020).

At the heart of DCN V2 is a more powerful cross network engineered to learn solid and wise-degree feature interactions (Quadratic Terms) accurately. DCN V2 differs from traditional models, where whitespace-based feature crosses are obtained mainly with deep neural networks, by introducing a multiple low-rank architecture. This boosts the model's expressiveness while keeping it computationally efficient. This allows the model to learn both explicit feature interactions through cross layers and implicit characteristic interplay among deep networks, providing a holistic view of the data (Wang et al., 2020).

Figure 13 illustrates the two ways to combine the cross and deep networks. In the stacked structure (Figure 13a), the cross network's output is fed into the deep network. In the parallel

structure (Figure 13b), both networks operate in parallel, and their outputs are combined (Ferrari Dacrema et al., 2019).

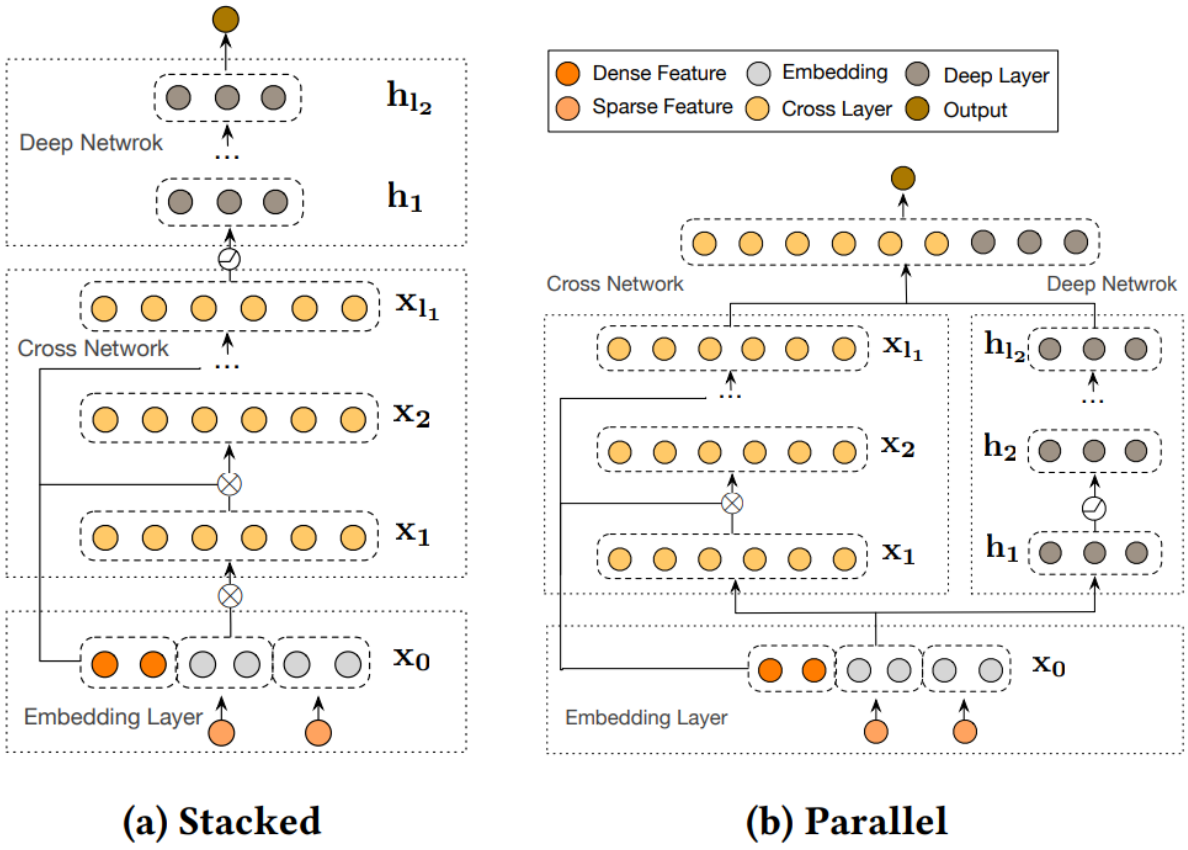

Figure 13 - Visualization of DCN-V2 (Wang et al., 2020).

DCN V2 is a major upgrade for learning-to-rank systems, particularly those that process web-scale data. It can learn the interaction of both explicit and implicit features, which is good for ranking use cases.

3.11 AUTOINT: AUTOMATIC FEATURE INTERACTION LEARNING

AutoInt is a novel neural network architecture for click-through rate (CTR) prediction in online advertising and recommender systems, built on higher-order feature interactions. CTR prediction is the task of predicting whether a user will click on an ad or item, and therefore plays a fundamental role in revenue for online platforms. Traditional methods have limitations, such as the sparsity and high dimensionality of input features (e.g., user IDs), which may include incorrect or categorical data like item categories. Furthermore, predicting

high-order combinatorial feature interactions—complex interaction effects in which multiple input features need to be captured simultaneously—adds further challenges.

To overcome these challenges, AutoInt introduces a novel method based on the idea of a multi-head self-attentive neural network. This network learns and models high-order interactions automatically, without requiring hand-crafted feature engineering by domain experts. AutoInt represents both numerical and categorical features in a joint low-dimensional embedding space, allowing powerful feature interactions by effectively using the available attention weights across them. Song et al. (2019) designed a multi-head self-attention mechanism for the model to learn important feature interactions at different layers, allowing this method to extract more concrete and subtle characteristics of various data properties.

The key ingredient of AutoInt is its interactions layer, which employs a multi-head self-attention mechanism. This enables every feature to interact with all the rest, linking relevant features that together generate high-order combinatorial properties (Song et al., 2019).

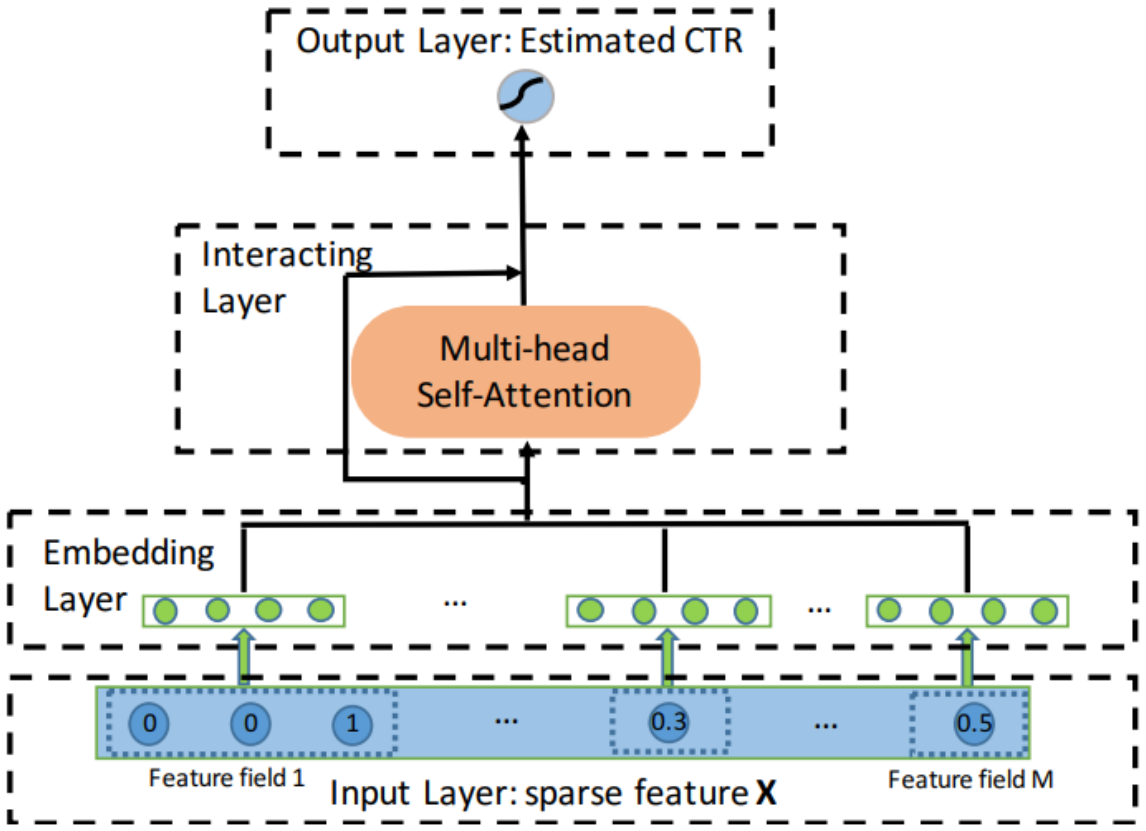

Figure 14 - Overview of the AutoInt Model (Song et al., 2019).

Here is a description of the model architecture for Click-Through Rate (CTR) prediction, including its AutoInt Figure 14. Starting from the input layer that reads in a sparse feature vector X –with elements of it representing each user or item. The embedding layer further abstracts these features and converts the high-dimensional sparse data into low-dimensional vectors. The stacked interacting layer at the heart of this model leverages multi-head self-attention to impose structured high-order feature interactions (Qu et al., 2016).

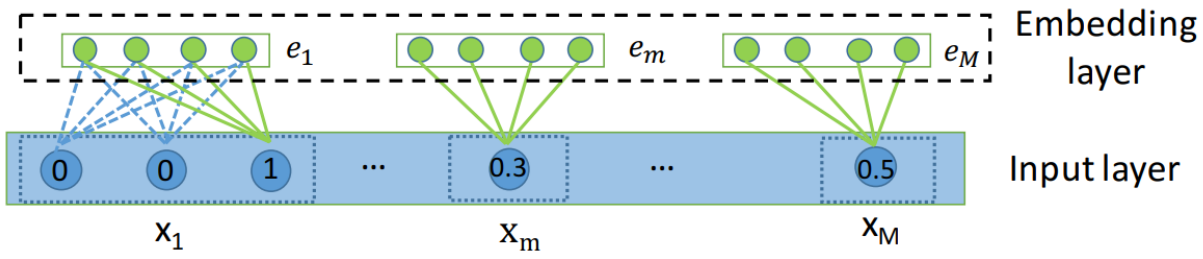

Figure 15 - Input and Embedding Layer (Song et al., 2019).

More specifically, the way the Input and Embedding Layer is used by AutoInt can be described as Figure 15. The figure shows low-dimensional dense vectors to encode the

fields, providing meaningful combinations of features in further layers for types and places (Song et al., 2019).

AutoInt is particularly efficient with large datasets. The model is end-to-end trainable, so it scales nicely with data size. Its capabilities to deal with complex, high-dimensional data and give interpretable insights regarding characteristic interactions make it a strong adjunct for boosting the performance as well as reliability of recommender systems (Qu et al. 2016).

3.12 Feature extraction from data

This paper presents a method for creating deep learning models that can predict stock trends and which are designed specifically to learn short-term spatial features from raw market data using Convolutional Neural Networks (CNN). We adopt the following three-step methodology: (1) construction of a market matrix from financial time series; (2) feature extraction using CNN to learn pruning spatial properties and PCA for further dimensionality reduction. By concatenating the outputs of both reduced PCA components and CNN model as features to train predictive models, our methods capture non-linear relationships inside data finally, which prove the great performance on processing high-dimensional financial time series trading costs (Chen et al., 2019).

Stock prices are extremely volatile in nature and various associated factors make the task even more complicated. These must be accommodated for within the model as traditional statistical techniques are often inadequate, capturing neither these nonlinear effects nor dynamics. Non-linear Relationships can be modeled using the latest Machine learning and Deep Learning techniques. We extend the dual feature extraction scheme introduced by Chen et al. in this paper. A CNN (Idrees et al., 2019) performed better in predicting short-term future power demand while PCA improved predictive performance by reducing the dimensionality of input feature space.

We then preprocess this data to make sure that the financials are all in a numeric format and also take care of missing values. Select the closing price and other technical indicators like Moving Average (MA), Relative Strength Index (RSI), Moving Average Convergence/Divergence (MACD), Stochastic Oscillator (Stoch_K, Stoch_D), and Williams

%R (WPR). A series of preprocessing steps are performed to prepare the data for our model. Feature selection pays attention to market data, which are Close, SMB, HML, LIQ, MOM, MA5, MA10, MA20, BIAS5, BIAS10, RSI6, RSI12, Stoch_K, Stoch_D, MACD, Signal, and WPR.

Standardizing the features, in this case, applying min-max scaling of its values (scales all feature values within a range, typically [0-1]). This means that you will need more data cleaning, deleting every row where we have NaN or infinite values to guarantee a complete and good working dataset. It also preserves the compatibility of your model by assuming that all variables are already numeric. We utilize a Convolutional Neural Network (CNN) for calculating spatial features in the preprocessed market matrix representation example, with rows as features and columns representing time points. In the CNN architecture, various convolutional layers (e.g., 1×3 and 1×5) are utilized to extract multi-level spatial features. The max pooling layers account for reducing feature maps produced.

The flattening layer creates an input form needed. The CNN model is trained with 10 days of data to predict the close price for a single day ahead. Regularization techniques (Chen et al., 2019) such as L2 regularization and dropout are employed to avoid overfitting, helping the model generalize well, which generally improves performance.

PCA is used to decrease the number of features that were extracted from CNN because of its high dimensionality while maintaining 99.9% variance. The CNN features are then flattened into a 2D matrix, and we follow with PCA to reduce the dimensionality, which gives us some principal components as input in our predictive models (Chen et al., 2019).

The features, on which PCA had been applied to reduce the dimensions, are combined with the target variable final data frame (closing price). To allow the model to generalize well on unseen data, this dataset is further divided into training and testing sets. Train various models on PCA components. These models are evaluated using a performance measure known as Root Mean Square Error (RMSE). Summary: The CNN-PCA algorithm provides an improvement over traditional methods with results indicating higher predictive power. The ability to perform well in various market conditions and temporal variations is witnessed, evidencing that the model can be generalized (Idrees et al.).

In this paper, we proposed a robust stock price trend prediction strategy by leveraging the ground-truth knowledge under a public environment which incorporated both CNN (for the

fine-grained spatial feature aggregation in short-term outliers) and the PCA algorithm for dimension reduction. Inspired by the work of Chen et al., a feature extraction methodology improves the predictive power of our model by making it more capable of capturing complicated market dynamics. In this study, we encode the traditional methods' results to represent them with principal components (PCs) for data mining and optimize model predictions by employing advanced machine learning to mitigate the limitations of others. The proposed feature mechanism model yields substantial improvements over the best existing methods, achieving superior generalizability and market forecast accuracy.

4. Feature extraction with CNN

We focused on predicting stock price trends using a Convolutional Neural Network (CNN) to extract short-term spatial features from market data. The methodology involves transforming the financial time series into a market matrix, applying a CNN to extract multi-level spatial features, and subsequently using Principal Component Analysis (PCA) for dimensionality reduction. The CNN model captures non-linear relationships within the data, and the reduced PCA components are used to train predictive models, demonstrating the effectiveness of our approach in handling high-dimensional financial time series data (Chen et al., 2019).

Predicting stock prices is a complex task due to the inherent volatility and multitude of influencing factors. Traditional statistical methods often fall short due to their linear assumptions and inability to capture intricate market dynamics. Recent advances in machine learning and deep learning offer new avenues for modeling these non-linear relationships. This paper builds upon the dual feature extraction methods highlighted by Chen et al. (2019), utilizing a CNN for short-term spatial feature extraction and PCA for dimensionality reduction to enhance predictive accuracy (Idrees et al., 2019).

The raw financial data undergoes preprocessing to ensure numeric consistency and address missing values. Key features such as the closing price and various technical indicators, including Moving Average (MA), Relative Strength Index (RSI), Moving Average Convergence/Divergence (MACD), Stochastic Oscillator (Stoch_K and Stoch_D), and Williams %R (WPR), are selected. To prepare the data for the model, a series of preprocessing steps are undertaken. Feature selection focuses on market data, including Close, SMB, HML, LIQ, MOM, MA5, MA10, MA20, BIAS5, BIAS10, RSI6, RSI12, Stoch_K, Stoch_D, MACD, Signal, and WPR. Min-max scaling is applied to standardize the features, ensuring they fall within a specific range, typically 0 to 1. Data cleaning involves handling NaN and infinite values by filtering out invalid rows to ensure the dataset is complete and accurate. Additionally, ensuring all features are numeric maintains compatibility with the model.

A Convolutional Neural Network (CNN) is employed to extract spatial features from the preprocessed market matrix, where each row represents a feature, and each column represents a time point. The CNN architecture includes multiple convolutional layers with

varying kernel sizes (e.g., 1×3, 1×5) to capture multi-level spatial features. Max pooling layers are incorporated to reduce the size of feature maps and prevent overfitting, while a flattening layer transforms the extracted features into a suitable format for further processing. The CNN model is trained to predict the next day's closing price using data from the past ten days. Regularization techniques such as L2 regularization and dropout are applied to prevent overfitting and enhance the model's generalization capability (Chen et al., 2019).

To address the high dimensionality of the CNN-extracted features, Principal Component Analysis (PCA) is applied to reduce the number of features while retaining 99.9% of the variance. The extracted CNN features are flattened into a 2D matrix, and PCA is performed to reduce dimensionality, resulting in a set of principal components that serve as inputs for the predictive models (Chen et al., 2019).

The PCA-reduced features are combined with the target variable (closing price) to form the final dataset. This dataset is split into training and testing sets to ensure the model's ability to generalize across unseen data. Various models are trained using the PCA components. Evaluation metrics such as Root Mean Square Error (RMSE) are used to assess the performance of these models. The CNN-PCA approach outperforms traditional methods, demonstrating superior predictive accuracy. Robustness to different market conditions and temporal variations is observed, indicating the model's generalization ability (Idrees et al., 2019).

We present a robust approach to stock price trend prediction by combining CNN for short-term spatial feature extraction with PCA for dimensionality reduction. The feature extraction methodology, inspired by the work of Chen et al. (2019), enhances the model's ability to capture complex market dynamics, leading to improved predictive performance. Our proposed approach addresses the limitations of traditional methods by effectively mining underlying market information and optimizing model prediction results through advanced machine learning techniques. This feature mechanism model provides a significant improvement over existing state-of-the-art methods, showcasing better generalization ability and market forecasting accuracy.

5. Results & Discussion

In this chapter, we present results of evaluating models on both train and test set and discuss the performance of each model on the test set. The main metric for comparing model performance is RMSE but we also consider MAE as well as other interesting plots like the residual plot. This section consists of two main parts, the first part discussing the results of models that are trained and tested on the TITLON dataset before feature extraction and the second part discussing the results after using extracted features made by CNN to train the same models. Our main goal is to show that feature extraction can improve the performance of deep learning models and GBDT models significantly.

5.1 Results before feature extraction

5.1.1 Results before feature extraction – deep learning models

In the Table 2, you can see RMSE and MAE of six deep learning models. The metrics are obtained from evaluating models on the test set (unseen data).

	SNN	Grownet	DCNv2	AutoInt	MLP	ResNet	FT-Transformer
RMSE	112.6	111.8	111.83	116	112.6	111.8	112.4
MAE	72.1	72.2	72.2	74.2	69.0	71.2	70.8

Table 2 – Deep learning models results before CNN feature extraction

Let's analyse the table in more detail.

Root Mean Square Error (RMSE)

The RMSE is simply the root mean square of differences between predicted values by a model and observed values. Lower RMSE values indicate better model performance since they show the model's predictions are closer to actual values.

The Self-Normalizing Network (SNN) that has an RMSE of 112.6 is one of the better performing models. The SNN was designed to facilitate good performance with deep architectures and big data while maintaining its inherent normalization across the network.

Grownnet (111.8), **DCNv2** (111.83): They both have similar RMSE values, outperforming SNN just a little bit. This implies that these models could be very good at capturing multiple connections within stock market data. It is possible that they can be using similar features which explains their similarity in performance.

It is with an RMSE of 116 that **AutoInt** gives the highest of all the models, thus indicating that it is worst in predicting the stock prices as opposed to others. Whereas the AutoInt models usually are very good at feature interaction capturing, in this case, maybe it has been overfitting or has failed to capture the correct relationships within the data.

On the other hand, the **MLP** does well but is not among the best models. It has a higher RMSE as compared to **Grownnet**, and DCNv2 but still does better than AutoInt. This could mean that although MLPs are able to model complex relationships, they might require more fine-tuning or structural changes to be as competitive as other highly specialized networks.

On the other side, **ResNet** structured itself into deeper form with skip connections and went on to perform well, returning an RMSE of 111.8 and therefore placing itself in very highly competitive models in this study. Probably skip connections assist the model in staying off various issues like a vanishing gradient; hence, issues are avoided and optimization and predictive accuracy go up.

FT-Transformer: 112.4. the FT-Transformer is quite good but marginally worse than either ResNet or Grownnet. It has normally been robust with respect to sequential dependencies, but this marginally worse RMSE may point out some more space for improvement, such as better hyperparameter tuning or changes in attention.

Mean Absolute Error (MAE)

MAE measures the average magnitude that errors in a set of predictions have, disregarding the direction. Thus, MAE provides a direct set of predictions in terms of how accurate the predictions are.

SNN, Grownnet, DCNv2: The predicted value by these models is very identical to each other, with SNN and Grownnet both being 72.1 and 72.2, respectively. DCNv2 comes in with an MAE of 72.2, which means all three models are very homogeneous for the lowest average error in this metric. Closeness in the MAE and RMSE values for these models shows that they substantially reduce the magnitude of errors.

Again, as was the case for the RMSE, **AutoInt** has the highest MAE, which shows its struggles to make correct predictions based on the dataset. The higher MAE indicates that the errors of AutoInt are mainly larger in size on average, probably due to the model's sensitivity to outliers or overfitting to noise in data.

In this respect, an **MLP** with an MAE of 69.0 has the smallest MAE and can thus be considered to produce the smallest average errors among all models. Again, this may imply that although the MLP possibly could not predict extreme values too accurately, this model is more stable and reliable in its predictions, making fewer large errors.

ResNet, 71.2 **FT-Transformer**, 70.8: Both models have MAE values slightly better than that of the SNN, Grownet, and DCNv2. ResNet with MAE of 71.2, which is in line with its very good RMSE performance, indicating a good balance toward the minimization of both large and small errors. The MAE of the FT-Transformer is 70.8, indicating that despite the higher RMSE, the model still manages to hold an average error relatively low, most likely thanks to its sequential modeling capabilities.

Key Observations from deep learning models

Best Overall Model: ResNet is a model with the lowest RMSE and has a competitive MAE among the models. Its deep architecture with skip connections seems to effectively capture the complexities in the stock price data.

Consistency Model: SNN, Grownnet, and DCNv2, are very much consistent in their performance, either in RMSE or MAE. Such consistency might make them more reliable choices, particularly when stability in prediction is important.

MLP's Robustness: The MLP does not possess the lowest RMSE but does have the lowest MAE; therefore, this may place it as one of the strongest models in terms of averaging out

errors. That makes MLP a good candidate in cases when one wants to guarantee fewer large prediction errors.

This, in turn, is the weak counterpart of the performance observed by the AutoInt model on this dataset, which would not really be the right track to capture feature interaction in the dataset. Additional tuning or a different model architecture might be necessary to get better performance.

Potential for Transformers: The FT-transformer model, while not reaching the top like ResNet, is promising because it can handle the sequential data. This points to the fact that it will give a better performance once optimized further.

In figure 16 you can see RMSE and MAE of the SNN model in each epoch. All other models mentioned in Table 2 showed the same pattern. Here are some interesting observations from the plots.

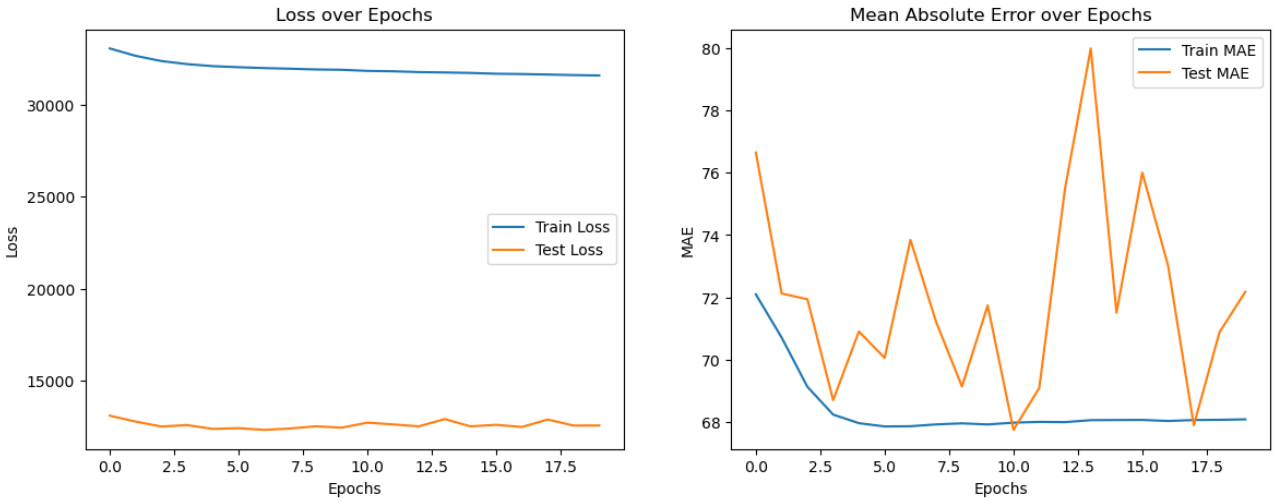

Figure 16 – SNN model training and test loss over 20 epochs.

Robustness to overfitting: The low-test loss and no increasing trend in test loss, shows good regularization in these models and hence no overfitting. This becomes a very common trait in financial data models since overfitting may lead to poor predictive performance when applied in real-world stock market situations.

Handling Volatility: Based on spikes in the test MAE, these models sometimes show problems with parts of the test data—likely due to high volatility periods. In any case, that they can recover from these spikes to maintain a low MAE does show that the models are robust and able to adjust after making a less accurate prediction. In stock price prediction, this resiliency will be very valuable, as market conditions can be very unpredictable.

Practical Implications: Based on the fact that models are pretty stable in terms of loss, together with recovering well from possible fluctuations in MAE, such models reveal good prospects for real-time stock price prediction. The accuracy and robustness presented by the models make them the ideal ones for practical applications in financial forecasting, with required performance in a variety of market conditions.

5.1.2 Results before feature extraction – GBDT models

In table 3, the results of two GBDT models that we used in this study are shown.

	XgBoost	CatBoost
RMSE	113	111.2
MAE	69.4	69.4

Table 3 - GBDT models results before CNN feature extraction

Comparing Performance of GBDT Models

CatBoost outperforms **XGBoost** at the lowest RMSE. This is probably because of the existence of a specialized algorithm in CatBoost that efficiently deals with the categorical features that are common in any financial dataset.

Both models turn out to perform the same with regard to MAE, suggesting that on average, predictions are similarly close to real values. This equality in the value of MAE suggests that the slightly better result in RMSE for CatBoost could be because of fewer bigger errors, which CatBoost is able to better switch.

In figure 17 you can see two plots from XGBoost model. The left plot is actual prices versus predicted prices by the model, and the right plot is the residual plot. There are some interesting insights in both plots.

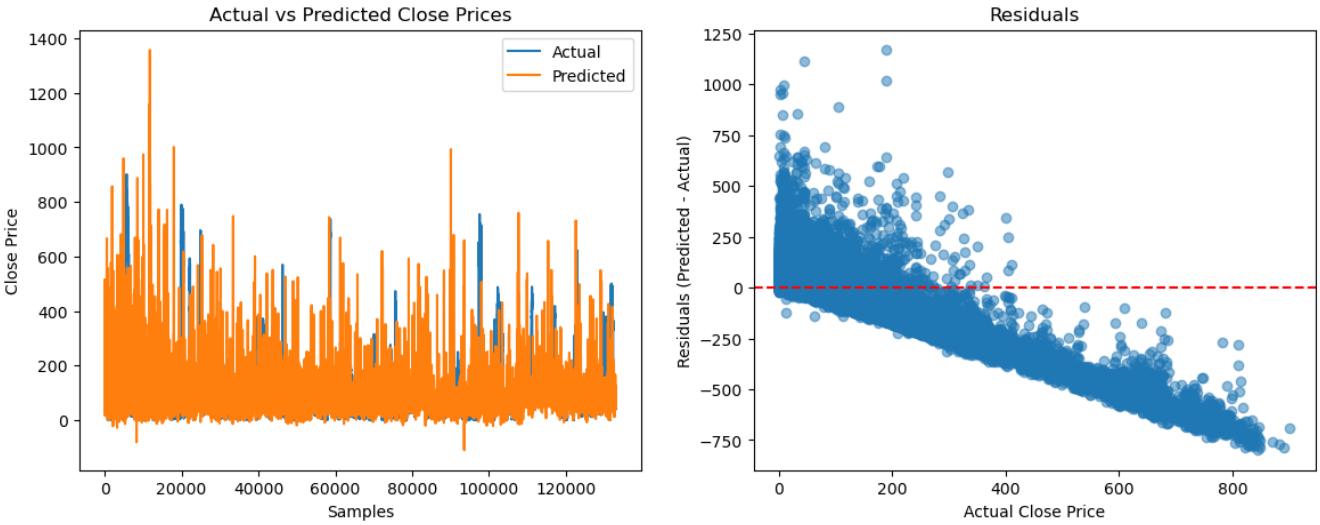

Figure 17- actual prices versus predicted prices - the residual plot

Left Plot: Actual vs. Predicted Close Prices

Overall Fit:

The orange line displays the predicted close prices, and the blue line plots the actual close prices. There is an even lining up of the two lines, particularly in those regions with lower prices and less volatility. This means the model is generally good at picking up the overall trend and level of the stock prices.

Even so, at times it could still not manage to match the real prices exactly, evidenced by visible gaps between the blue and orange lines, where there is higher price volatility, such as spikes. It therefore shows that even though the model is generally good at predicting the trends, it might not pick up on extreme movements or changes in prices.

High and Low Prices:

It appears to be most accurate for the more frequent, lower and medium-range price points, where the real values are very close to the predicted ones. For larger price spikes, the

predicted values are less than the real prices, which could indicate that this model is actually more conservative or even unable to catch the dynamics in periods of fast price increase.

Right Plot: Residuals (Predicted – Actual)

Distribution of Residuals

This residual plot is the difference between the predicted and actual prices for the range of the actual prices. Ideally, residuals should be randomly distributed around the horizontal line that represents zero error, indicating that the model does not systematically overpredict or underpredict.

In this scatter plot, the residuals tend to bunch near zero, especially for lower actual close prices. This gathering may mean the model predictions are very good for the lower price ranges since there is not much deviation from the real values.

Systematic Bias:

There is a visible downward trend in the residuals as the actual close price increases. This may imply that, with an increase in actual price, the model tends to underpredict. The negative residuals in the higher price range suggest that the model has some systemic bias toward underestimation of the high prices, which could be a limitation for predicting high price values.

Magnitude of Mistakes:

We can notice that residuals are more spread out for higher prices, which again goes in line with the finding from the first plot: the model does worse for correctly predicting higher prices. Also, a wider spread in residuals in this range can indicate increased difficulty the model is having while working on more volatile or less frequent price points.

Improvement in Model: Residuals are found to be constant. There are some possibilities of improvement in the model, either by including more features or by tuning its complexity and hyperparameters to fit well on this higher price range. This bias may be corrected for better performance, especially for higher-priced stocks or time frames.

Comparison with Deep Learning Models

RMSE Performance:

Comparing these GBDT models with the deep learning models illustrated above, we can observe that CatBoost competes with a very valid RMSE of 111.2, beating out all models like MLP and AutoInt, and ResNet. The RMSE of XGBoost, 113, is considerably above most deep learning models, indicating that while the former is competitive, it likely misses several complex dependencies that a deep learning model can handle.

MAE Performance:

The reported MAE of 69.4 on both XGBoost and CatBoost outperforms many deep learning models, like SNN and ResNet, that are around 71-72. This proves that these GBDT models perform very well at minimizing average prediction errors, hence they are reliable models if the goal is to obtain the least average error.

Model Selection Considerations

CatBoost and the other GBDT-based models are preferable for applications that require high computational efficiency and ease of implementation, due to their confirmed strong MAE performance and relatively competitive RMSE. These models are best for using in circumstance with smaller datasets or when interpretability is the key in such circumstances. On the other hand, deep learning models may be applied when reaching the smallest possible RMSE is wanted, and in the case of dealing with extremely complex patterns in very big datasets.

5.2 Results after CNN feature extraction

5.2.1 Results after CNN feature extraction – deep learning models

In table 4, the results of evaluation deep learning models on the test set that are trained with tensors obtained from CNN are shown. There are many interesting observations in these results. Metrics are shown in two states: obtained from scaled data and unscaled data.

	SNN	Grownet	DCNv2	AutoInt	MLP	ResNet	FT-Transformer
RMSE	8.1	73.7	468587200.0	8.01	7.9	62.3	80.1
MAE	2.05	35.9	289063488	1.6	1.5	38.9	141
RMSE Scaled	8.05	52.2	8.1	8.6	8.1	264.7	54.3
MAE Scaled	1.8	31.3	1.7	3.8	1.9	264.5	28.3

Table 4 – Deep learning models results after CNN feature extraction

Performance Overview Post-CNN feature extraction

Unscaled Metrics Analysis

RMSE (Root Mean Square Error):

SNN: The RMSE improved dramatically from 112.6 to 8.1, showing a tremendous improvement in prediction accuracy.

Grownet: With an RMSE of 73.7, even though this is still higher compared to some other models, performance has vastly improved from its previous RMSE of 111.8.

DCNv2: It has an RMSE of 468587200.0. This is a very high value, indicating likely some computational mistake or misconfiguration at some step of the feature extraction/evaluation. In the next section we will see the amazing effect of scaling on the result of this model.

AutoInt: The RMSE dropped from 116 to 8.01, showing a dramatic improvement.

MLP: Improved significantly from 112.6 to 7.9.

ResNet: That's a good drop in the RMSE from 111.8 to 62.3, and it still stays higher compared to most of the other models.

FT-Transformer: It has shown a fair drop in the RMSE from 112.4 to 80.1. This is still higher compared to most of the models after feature extraction.

MAE—Mean Absolute Error :

SNN: From an MAE of 72.1, it improved to 2.05, improving greatly in prediction accuracy.

Grownet: The MAE dropped drastically from 72.2 to 35.9, but it is still much worse than the best models.

DCNv2: A very high MAE of 289063488 means there is likely to be a huge mistake or outlier in the evaluation. It is effected by outliers that will be fixed by scaling the data.

AutoInt: from 74.2 to 1.6, the best improvements among all models.

MLP: from 69.0 to 1.5, being at the top of the leaderboard in this measure.

ResNet: The MAE decreased from 71.2 to 38.9, one of the bigger drops but still higher than the top-performing models.

FT-Transformer: From 70.8 to 141, it improved a bit and was at that point still incomparable to the best models.

Scaled Metrics Analysis

The metrics, if scaled, will further elucidate the relative improvements in performance. Notably, SNN retains very strong performance with the scaled RMSE at 8.05 and the scaled MAE at 1.8. Clearly, it holds on to its place as one of the best performers after the feature extraction.

Grownet: While improvement is provided, the scaled metrics remain worse than the others with an RMSE of 52.2 and MAE of 31.3.

DCNv2: The scaled metrics proves that the previous high RMSE and MAE were result of outliers and volatile prices. As you can see after scaling the result for this model become amazing and beats the best models; RMSE 8.1, MAE 1.7.

AutoInt: Still very strong; on scaled RMSE, it comes in at 8.6, and on MAE, it comes in at 3.8, making this one of the better models after feature extraction.

MLP: Performs really well: scaled RMSE comes in at 8.1, MAE at 1.9.

ResNet: While improved before scaling, the metrics for scaled RMSE and MAE of 264.7 and 264.5, respectively, show its poor relative performance to other models with CNN-transformed data after scaling. This can be due to the complex structure of ResNet model that performs better with the original data.

FT-Transformer: The metric values of scaled RMSE 54.3 and MAE 28.3 explain that while it improves, the FT-Transformer still fails to be one of the top performers.

Comparison with Pre-CNN feature extraction Results

Comparison of the results for CNN-transformed models with the performance of the original models returns the following observations:

Huge improvements: almost all of them significantly reduced RMSE and MAE, thereby truly reflecting the effect of CNN feature extraction that enabled most of them to learn much better from the TITLON dataset. In particular, the SNN, AutoInt, and MLP models presented the largest reduction in error metrics.

Grownnet, FT-Transformer and ResNet: Their errors decreased, but they still stand higher than the errors obtained by the best-performing models. This indicates that these models benefit sufficiently from the CNN feature extraction and that the transformed data requires more tuning and adjustment to capture intricate patterns in the data.

DCNv2 performance plots

Figure 18 – DCNv2 train-test loss before scaling

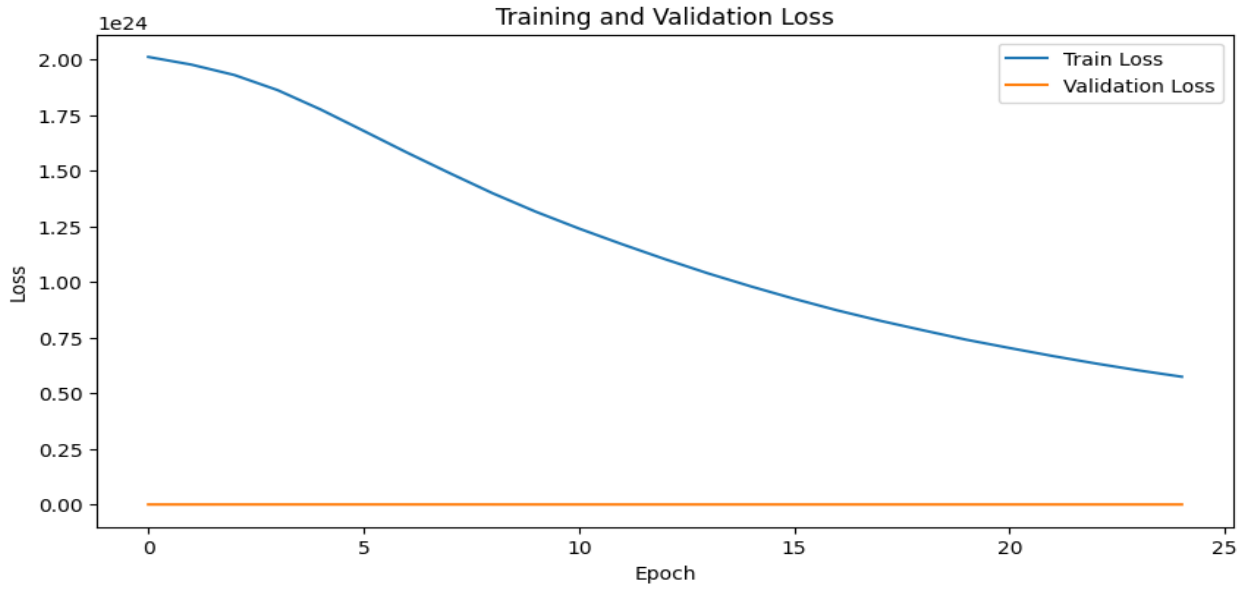

Figure 19 – DCNv2 train-test loss after scaling

Plot of Loss Before Scaling, figure 18:

Training Loss: It is certainly noticed here that the loss is pretty high in the beginning and very rapidly lowers with each passing epoch. This, quite obviously, signifies that the model learns at the very fast pace and does not follow a normal training process. Large magnitude of loss, however, it usually means that the point of the model is making far-off predictions from the real set of values; probably due to the unscaled input data.

During training, the validation set loss remains very close to zero and completely flat. This usually is a very serious problem: probably the model is overfitting the training data.

Plot of Loss after Scaling, figure 19:

Training Loss: Training loss starts high and gradually decreases over epochs before flattening. Much more of a stable and decent process for training—models learn at a reasonable scale and converge at the speed of a solution. It simply means that scaling is rather important since all the input features are put at the same scale so that no single feature has a very large range of values that could destabilize training.

Validation Loss: The trend for validation loss after scaling is very much the same as that of the validation loss before scaling, very low and almost flat. This was already indicated, that the model was not overfitting. These two curves are very well aligned from the post-scaling stage and give an indication of how the model can generalize over unseen data very well.

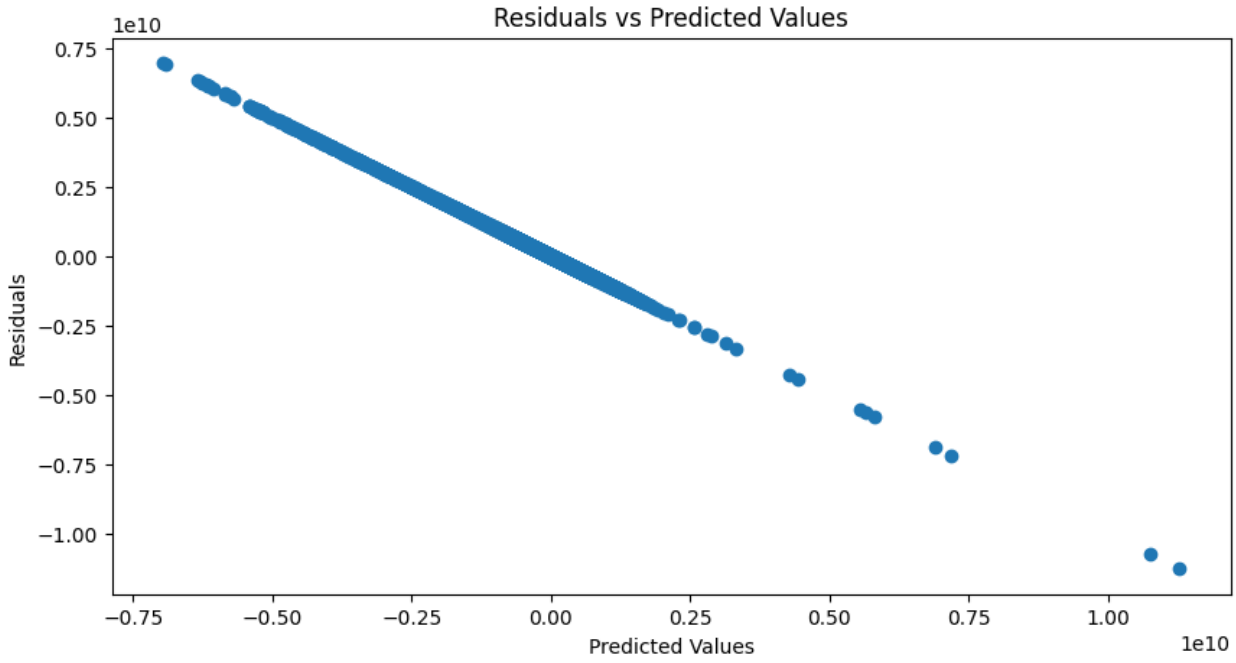

Figure 20 – DCNv2 residual plot before scaling

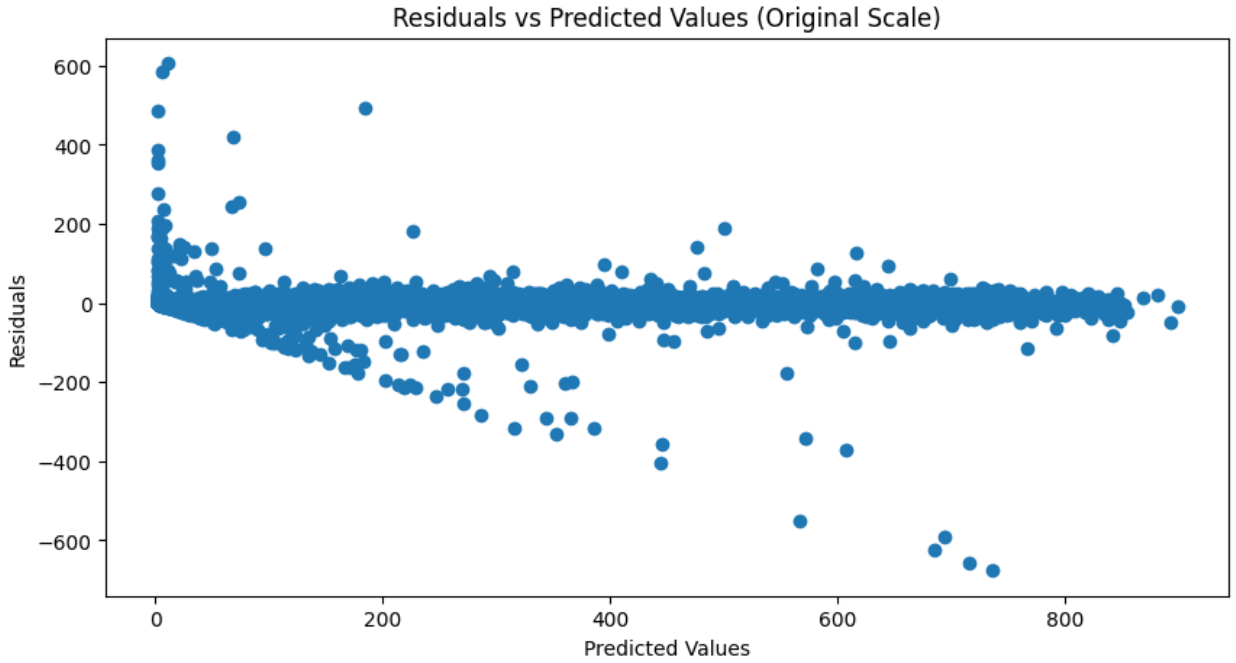

Figure 21– DCNv2 residual plot after scaling

Residuals vs. Fitted Values Before and After Scaling

Before Scaling, figure 20:

Residual Pattern:
The residuals before scaling display a very nonrandom straight-line pattern instead of what is wanted, that is, a random scatter of points about the horizontal axis where residual = 0.
This clearly linear pattern of residuals says that the model may be biased in a way such that it underpredicts or overpredicts certain ranges of the target variable. This might occur because the features fed into the model would possibly be on a different scale, hence the result in systematic errors.

Implications for Residual Non-Randomness:
Non-random residuals indicate that there is something wrong with the model; it's not modeling the underlying relationships between variables and there is some information left in the data that the model in unable to capture. The model has systematic errors, which seriously diminishes its effectiveness in prediction on real-world data.

After Scaling, figure 21:

Residuals Pattern:
After scaling, the residuals are much more randomly located around the horizontal axis, which is ideal. This means it has no discernible pattern of residuals; that is, the errors of the model are now much more randomly located. This puts the residuals into a much more even spread across different ranges of the predicted values, further indicating that the model now handles these different scales of input data more effectively.

Implications of Improved Residuals:
This would be preferable since residuals, after scaling, do contain randomness and signal that the model is making mistakes, but not systematically. In this case, it would suggest that

we are capturing relationships more fundamentally in the data, hence reducing the likelihood of missing these patterns and information. The improved residuals increase the reliability of the model in making a prediction. It is because of the less vulnerability of this model to systematic errors that is makes this tool very robust in financial forecasting tasks.

5.2.2 Results after feature extraction – GBDT models

	XGBoost	CatBoost
RMSE	15.01	16.01
MAE	2.84	3.8
RMSE Scaled	14.9	16.02
MAE Scaled	2.8	3.8

Table 5 – GBDT models results after CNN feature extraction

Comparative Analysis of Models Pre- and Post-CNN feature extraction:

The MAE improvement has been seen to be extremely different for both models after the CNN feature extraction, which is from 69.4 to 2.84 on the XGBoost model and from 69.4 to 3.8 on CatBoost. Hence, in general, the improvement rate is so dramatic that it shows the CNN feature extraction actually increases the capabilities of the models to predict with a smaller error on average.

Improvement in RMSE: The RMSE values in both the models decreased, which means predictive accuracy increased. For example in the XGBoost model, it decreased from 113 to 15.01, and in CatBoost, it decreased from 111.2 to 16.01.

Comparison to Deep Learning Models after CNN feature extraction:

Performance of Deep Learning: Among the deep learning methods that had already been looked at, SNN, AutoInt, and MLP improved drastically, falling to RMSE values of about 8-9, and the MAE also improved by an equivalent extent.

Comparing the MAE: After feature extraction, GBDT models—XGBoost and CatBoost—provided a significant decrease in MAE with values of 2.84 and 3.8, respectively. Improvements similar to those offered by some deep learning models, thus showing that the effectiveness of the CNN feature extraction worked very well for GBDT models too.

Comparing RMSE: The RMSE values for GBDT models after feature extraction, which are around 15–16, turn out to be slightly higher than that of the best deep learning models whose RMSE values have been of the order of 8–9. That means, while GBDT models did improve, deep learning models still seem to have an edge in terms of lesser deviations from actual values.

Some key insights

CNN feature extraction benefits: The addition of the CNN feature extraction comes with a lot of performance gain in the GBDT models as seen in the substantive decrease of both the RMSE and MAE that might mean that the models are now catching more of the complex relationship pattern within financial data. With these improvements, deep learning models still maintain an edge over GBDT models in RMSE, with these model types being more suitable for learning while reducing large deviations in the complex data sets such as TITLON.

Interpretations of XGBoost **figures**

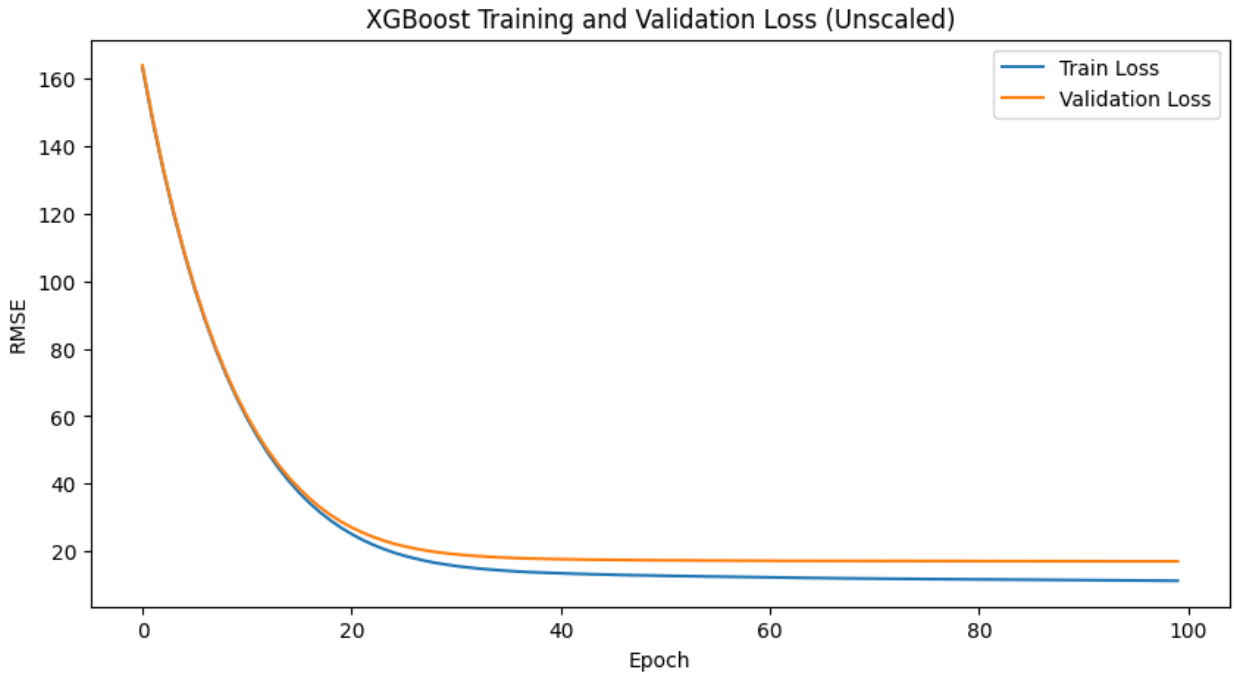

Figure 22 – XGBoost train-test loss after feature extraction

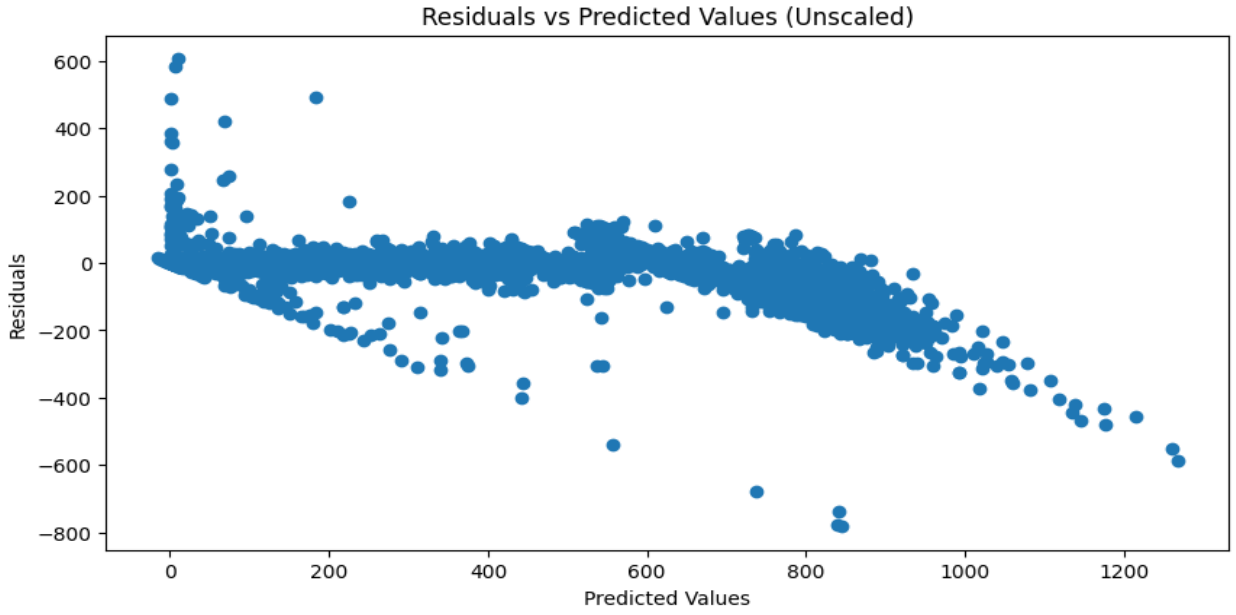

Figure 23 – XGBoost residual plot after feature extraction

Insights from figures 22 and 23.

The overall performance of this XGBoost model was good in the sense that there was a steep decrease and some form of stabilization for both the training and validation loss curves, using CNN-transformed data. This indicates that the model generalized quite well to the validation set without overfitting, which can be taken as a positive sign toward robustness.

Residual patterns: It is still observable on the residual plot that there are some biases of the predictions by the model, more towards the extremes of the predicted value range. This graph looks for improvement possibilities and points to either further tuning or more sophisticated feature extraction, which will be able to handle these systematic errors.

Comparison to Other Models: The performance of this XGBoost model relative to other models is rather good in terms of training and validation loss; however, the residuals show that there is room for improvement to actually benefit from the CNN feature extraction.

Bias in Predictions: The residual pattern indicates some bias in the model's predictions. There is increasing spread toward the lower predicted values, which may suggest some underestimation by the model; curvature at higher predicted values could quite possibly mean overestimation or at other values.

Therefore, these patterns would infer that overall, although the model goes well according to the loss curves, there may still exist a possibility for further tuning or additional feature engineering that can help bring down these systematic errors.

6. Conclusion

The results are summarized in Table 6, which compares numerous deep learning and GBDT models on the TITLON dataset, that represents stock price data from the Oslo Stock Exchange, with and without the CNN-based feature extraction. This section summarizes major conclusions drawn from the results.

	SNN	Grownet	DCNv2	AutoInt	MLP	ResNet	FT-Transformer	XGBoost	CatBoost
RMSE	112.6	111.8	111.83	116	112.6	111.8	112.4	113	111.2
MAE	72.1	72.2	72.2	74.2	69.0	71.2	70.8	69.4	69.4
RMSE Transformed	8.05	52.2	8.1	8.6	8.1	264.7	54.3	14.9	16.02
MAE Transformed	1.8	31.3	1.7	3.8	1.9	264.5	28.3	2.8	3.8

Table 6 – all results together

1. **Pre-CNN feature extraction Analysis**

SNN, Grownnet, and DCNv2: When we take a closer look at the table, the observed similarities in these models were RMSE values around 111-112 and MAE values ranging in between 72.1-72.2, suggesting they were relatively in conformity with each other. Through these similarities, it proposes consistently between these models.

The average general performance of ResNet and FT-Transformer is also related to that of SNN, giving RMSE values around 111-112 but slightly lower in MAE compared to SNN and Grownnet.

XGBoost and CatBoost: In Two GBDT-based views of XGBoost and CatBoost, XGBoost is neck-and-neck in results with deep learning models, returning the RMSE at about 113 and MAEs at 69.4. But CatBoost is obviously outperform all deep learning models with RMSE of 111.2. This indicates that there is no universal best model or group of models and depending on the data and feature selection methods one group of models can outperform the others.

2. Analysis after CNN feature extraction

When we look at the results of the metrics after CNN-based feature extraction, the drastic change in performances shows the necessity of feature extraction and transformation for model accuracy improvement:

SNN: The SNN showed a steep drop in both RMSE and MAE after applying feature extraction, where the RMSE went down to 8.05 and the MAE went down to 1.8. This tremendous improvement shows that CNN can further enhance the ability of representation of features so that SNN can capture more patterns in the data and detect the underlying complexity.

Grownnet and **DCNv2**: In comparison with the best models, Grownnet gave an RMSE of 52.2 and an MAE of 31.3. On the other side, DCNv2 gave a very low RMSE of 8.1 and an MAE of 1.7, indicating that this model was particularly benefited by CNN feature extraction. That might be due to the architecture itself, which provides better conditions for exploiting the enriched features.

AutoInt and **MLP**: Both models turned out significant improvements, with RMSE values at around 8.6 and 8.1, respectively. The MAE improved considerably enough to place both these models at par with other good-performing models after feature extraction.

While **ResNet** did not gain such a benefit from the CNN feature extraction, with their values of root mean squared errors increasing drastically, with RMSE of 264.7 which may suggest

possible overfitting or unsuitability with this new transformed feature space. This could imply that further tuning or model architectural changes might be necessary to utilize such CNN feature extraction fully.

XGBoost and **CatBoost**: The GBDT models also improved after feature extraction, with XGBoost now getting an RMSE of 14.9 and an MAE of 2.8, while CatBoost achieved an RMSE of 16.02 and an MAE of 3.8. While these improvements are quite significant, as compared to their performance before feature extraction, they remain slightly worse than the best deep learning models with respect to RMSE though competitive in MAE values.

3. Comparative Insights and Final Observations

Results indicate that the CNN-based feature extraction really made a huge difference in improving model performance, especially with regard to reducing RMSE and MAE for most models. Some of the key takeaways are as follows:

Comparatively better deep learning models: SNN, DCNv2, AutoInt, and MLP enjoyed maximum benefits from the CNN feature extraction, yielding the lowest RMSE and MAE values. In this respect, it highlights what is additionally gained when CNNs are combined with other deep learning models to extract complex features for financial data modeling in efforts geared toward better predictive performance.

Resilience of GBDT Models: Not as good as the best Deep Learning models, XGBoost and CatBoost significantly improved on MAE. This offers evidence that GBDT models can still be competitive when joined with powerful feature extraction, particularly for those problems at hand where average error minimization could be most important.

Challenges for Certain Architectures: Overall, the bad performance that ResNet and FT-Transformer give post- feature extraction does lead to the viewpoint that not all deep learning architectures will turn out to be well-suited for these CNN-based feature extraction. This indicates the need for a very careful model selection and tuning while applying such

complicated feature extraction or other methods, since some models need more advanced adjustments to adapt better to the transformed features.

4. Conclusion

Feature extraction is a key part of most predictive models, and this result is well displayed in the comprehensive analysis presented in Table 6 for a stock price prediction scenario over the TITLON dataset. The CNN-based feature extraction significantly improved most models, which were anyway quite good with the treatment of complex, high-dimensional data. Models such as SNN, DCNv2, and MLP emerged as the top performers, which further demonstrated the elements of deep learning in the effective feature extraction technique.

Such an inconsistent impact on models such as ResNet and FT-Transformer shows that the problem is about applying one-size-fits-all feature extraction across architectures. This should clearly attest to the necessary tailor-made approach with respect to model selection and model tuning so that each model is well set up to harness the benefits of feature extraction to the maximum.

To sum up the findings, the CNN feature extraction significantly enhanced the performance in prediction, but the selection of the architecture of the model and the careful tuning of the hyperparameters remained the things that are definitely vital for optimal performance. It will be possible for this study to be extended to the optimization of a variety of machine learning and deep learning approaches in financial forecasting.

References

Bjorvatn, Kjetil, and Tina Søreide. 2003. Corruption and market reform. In *Discussion Paper*. Bergen: NHH Department of economics.

Boye, Knut, and Geir Severinsen. 1996. *Finansielle emner*. 10. utg. Oslo: Cappelen akademisk forl.

Lipton, Z. C. (2016). The mythos of model interpretability. *Communications of the ACM, 61*(10), 36–43. https://doi.org/10.48550/arXiv.1606.03490.

Hyndman, R. J., & Athanasopoulos, G. (2018). *Forecasting: Principles and Practice* (2nd ed.). OTexts. Retrieved from: https://otexts.com/fpp2/.

W. Song, C. Shi, Z. Xiao, Z. Duan, Y. Xu, M. Zhang, and J. Tang. AutoInt: Automatic feature interaction learning via self-attentive neural networks. In CIKM, 2019.

R. Wang, R. Shivanna, D. Z. Cheng, S. Jain, D. Lin, L. Hong, and E. H. Chi. DCN v2: Improved deep & cross network and practical lessons for web-scale learning to rank systems. arXiv preprint arXiv:2008.13535v2, 2020.

S. Popov, S. Morozov, and A. Babenko. Neural oblivious decision ensembles for deep learning on tabular data. In ICLR, 2020.

S. Badirli, X. Liu, Z. Xing, A. Bhowmik, K. Doan, and S. S. Keerthi. Gradient boosting neural networks: Grownet. arXiv preprint arXiv:2002.07971v2, 2020.

Gorishniy, Y., Rubachev, I., Khrulkov, V., & Babenko, A. (2021). Revisiting deep learning models for tabular data. In Advances in Neural Information Processing Systems, 34, 18932–18943.

Brealey, Richard A., and Stewart C. Myers. 2003. *Principles of Corporate Finance*. 7th ed. Boston, Mass.: McGraw-Hill/Irwin.

Copeland, Thomas E., Tim Koller, and Jack Murrin. 2000. *Valuation : measuring and managing the value of companies*. 3rd ed, *Wiley finance*. New York: Wiley.

Gabrielsen, Hanne Cathrin, and Turid Moldenæs. 2002. Kvalitetssikring og etisk regnskap: en nyinstitusjonelt perspektiv. *Beta. Tidsskrift for bedriftsøkonomi* (1):28-50.

Lillestøl, Jostein. 2002. Value at risk: sett med en statistikers øyne. In *Essays on Uncertainty: Festskrift til Steinar Ekerns 60-årsdag.*, edited by P. Bjerksund and Ø. Gjerde. Bergen: Norges handelshøyskole.

Sarker, I. H. (2021). Machine learning: Algorithms, real-world applications and research directions. *SN Computer Science, 2*(3), 160. https://doi.org/10.1007/s42979-021-00592-x.

Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The elements of statistical learning: Data mining, inference, and prediction* (2nd ed.). Springer. https://doi.org/10.1007/978-0-387-84858-7.

Chen, T., & Guestrin, C. (2016). XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 785–794). https://doi.org/10.1145/2939672.2939785.

Nelson, D. M., Pereira, A. C., & de Oliveira, R. A. (2017). *Stock market's price movement prediction with LSTM neural networks.* In *2017 International Joint Conference on Neural Networks (IJCNN)* (pp. 1419-1426). IEEE. https://doi.org/10.1109/IJCNN.2017.7966019

Keogh, E., & Mueen, A. (2017). Curse of dimensionality. In *Encyclopedia of Machine Learning and Data Mining* (pp. 314–315). Springer. https://doi.org/10.1007/978-1-4899-7687-1_192.

Tishby, N., & Zaslavsky, N. (2015). Deep learning and the information bottleneck principle. In *2015 IEEE Information Theory Workshop* (pp. 1–5). https://doi.org/10.1109/ITW.2015.7133169.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning.* MIT Press. https://www.deeplearningbook.org

Nguyen, K., Mai, T. N., Nguyen, H. A., and Nguyen, V. A. (2023). A computational model for predicting customer behaviors using transformer adapted with tabular features. *International Journal of Computational Intelligence Systems*, 16:128.

Cho, C.-H., Kang, J., and Cheon, H.-J. (2022). Online shopping hesitation. *CyberPsychol. Behav.*, 9(3), 261-274.

ResearchGate. (n.d.). The FT Transformer architecture. Retrieved from https://www.researchgate.net/figure/The-FT-Transformer-architecture-the-proposed-architecture-contains-two-major-blocks_fig1_372985818. Accessed: 13.02.2024.

Gupta, V. (2023). Wavelet transform and vector machines as emerging tools for computational medicine. J. Ambient. Intell. Humaniz. Comput., 14(4), 4595-4605.

S. O. Arik and T. Pfister. Tabnet: Attentive interpretable tabular learning. *arXiv*, 1908.07442v5, 2020.

Devlin, J., Chang, M., Lee, K., and Toutanova, K. (2018). BERT: Pre-training of deep bidirectional transformers for language understanding. *arXiv:1810.04805*.

Chen, J., Song, L., Wainwright, M. J., and Jordan, M. I. (2018). Learning to explain: An information-theoretic perspective on model interpretation. *arXiv:1802.07814*.

Ibrahim, M., Louie, M., Modares, C., and Paisley, J. W. (2019). Global explanations of neural networks: Mapping the landscape of predictions. *arXiv:1902.02384*.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *arXiv*, 1512.03385v1, 2015b.

Szegedy, C., Ioffe, S., Vanhoucke, V., and Alemi, A. Inception-v4, Inception-ResNet and the impact of residual connections on learning. *arXiv*, 1602.07261v2, 2016.

C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9, 2015.

K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

I. Sutskever, J. Martens, G. Dahl, and G. Hinton. On the importance of initialization and momentum in deep learning. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, volume 28, pages 1139–1147. JMLR Workshop and Conference Proceedings, May 2013.

T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama. Optuna: A next-generation hyperparameter optimization framework. In *KDD*, 2019.

I. Tolstikhin, N. Houlsby, A. Kolesnikov, L. Beyer, X. Zhai, T. Unterthiner, J. Yung, A. Steiner, D. Keysers, J. Uszkoreit, M. Lucic, and A. Dosovitskiy. MLP-Mixer: An all-MLP architecture for vision. *arXiv preprint arXiv:2105.01601v4*, 2021.

I. Bello, W. Fedus, X. Du, E. D. Cubuk, A. Srinivas, T.-Y. Lin, J. Shlens, and B. Zoph. Revisiting ResNets: Improved training and scaling strategies. *arXiv preprint arXiv:2103.07579*, 2021.

T. Chen, S. Singh, B. Taskar, and C. Guestrin. Efficient second-order gradient boosting for conditional random fields. In *Proceeding of 18th Artificial Intelligence and Statistics Conference (AISTATS'15)*, volume 1, 2015.

X. He, J. Pan, O. Jin, T. Xu, B. Liu, T. Shi, A. Atallah, R. Herbrich, S. Bowers, and J. Q. Candela. Practical lessons from predicting clicks on ads at Facebook. In *Proceedings of the Eighth International Workshop on Data Mining for Online Advertising, ADKDD'14*, 2014.

L. Prokhorenkova, G. Gusev, A. Vorobev, A. V. Dorogush, and A. Gulin. CatBoost: unbiased boosting with categorical features. In *NeurIPS*, 2018

M. Ferov and M. Modrý. Enhancing lambdamart using oblivious trees. *arXiv preprint arXiv:1609.05610*, 2016.

G. Klambauer, T. Unterthiner, A. Mayr, and S. Hochreiter. Self-normalizing neural networks. In NIPS, 2017.

Y. Bengio. Deep learning of representations: Looking forward. In *Proceedings of the First International Conference on Statistical Language and Speech Processing*, pages 1–37, Berlin, Heidelberg, 2013.

W. Bao, W.-S. Lai, C. Ma, X. Zhang, Z. Gao, and M.-H. Yang. Depth-aware video frame interpolation. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3698–3707, 2019.

G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

M. Ferrari Dacrema, P. Cremonesi, and D. Jannach. Are we really making much progress? A worrying analysis of recent neural recommendation approaches. In *Proceedings of the 13th ACM Conference on Recommender Systems*, pages 101–109, 2019.

W. Cheng, Y. Shen, and L. Huang. Adaptive factorization network: Learning adaptive-order feature interactions. *arXiv preprint arXiv:1909.03276*, 2019.

H.-T. Cheng, L. Koc, J. Harmsen, T. Shaked, T. Chandra, H. Aradhye, G. Anderson, G. Corrado, W. Chai, M. Ispir, et al. Wide & deep learning for recommender systems. *arXiv preprint arXiv:1606.07792*, 2016.

T. Chen, J. Lin, T. Lin, S. Han, C. Wang, and D. Zhou. Adaptive mixture of low-rank factorizations for compact neural modeling. 2018.

A. Beutel, P. Covington, S. Jain, C. Xu, J. Li, V. Gatto, and E. H. Chi. Latent cross: Making use of context in recurrent recommender systems. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, pages 46–54, 2018.

Y. Qu, H. Cai, K. Ren, W. Zhang, Y. Yu, Y. Wen, and J. Wang. Product-based neural networks for user response prediction. In *Data Mining (ICDM), 2016 IEEE 16th International Conference on*, pages 1149–1154. IEEE, 2016.

Y. Chen, W. Lin, and J. Z. Wang. A dual-attention-based stock price trend prediction model with dual features. *IEEE Access*, 2019.

S. M. Idrees, M. A. Alam, and P. Agarwal. A prediction approach for stock market volatility based on time series data. *IEEE Access*, vol. 7, pp. 17287–17298, 2019.

https://www.euronext.com/nb/markets/oslo

https://titlon.uit.no/