Prediction of Market Value Based on Changes in Capital and Asset Structure Using Convolutional Neural Networks

Kacper Sobczyk
ksobczyk@student.agh.edu.pl

May 2024

Abstract

The study examines the effectiveness of convolutional neural network (CNN) in identifying patterns within financial data to improve market value predictions. The model is trained on historical data from the Polish stock exchange (GPW) covering the period from 1998 to 2022 on a quarterly basis and evaluated on its predictive accuracy using Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and R-squared (R^2). Experiments summary, limitations, and suggestions for future research are also discussed.

1 Introduction

1.1 Motivation

The valuation of companies and their stocks is a critical aspect of financial markets, influencing investment decisions, corporate strategies, and economic policies. Accurate prediction of market value is essential for investors seeking to maximize returns and minimize risks. Traditional methods for predicting market value often rely on linear models and fundamental analysis, which may not fully capture the complexities of financial data. With the advent of advanced computational techniques and the availability of large datasets, machine learning approaches, particularly convolutional neural networks (CNNs), have shown promise in enhancing predictive accuracy by identifying intricate patterns within the data.

Convolutional neural networks, initially developed for image recognition tasks, have been successfully applied to various domains, including natural language processing, medical diagnosis, and now, financial forecasting. The strength of CNNs lies in their ability to detect spatial hierarchies in data, making them well-suited for analyzing time series and structured financial information. This paper explores the application of CNNs to predict market value based on changes in capital and asset structure, leveraging historical data from the Polish stock exchange. By transforming financial metrics into formats that can be processed by CNNs, we aim to enhance the accuracy and reliability of market value predictions. The motivation for this research stems from the limitations of traditional financial analysis methods and the potential of machine learning techniques to address these challenges. Traditional models often assume linear relationships and may overlook complex interactions within financial data. Furthermore, these models might not fully exploit the wealth of historical data available, leading to suboptimal predictive performance.

Machine learning, and CNNs in particular, offer a new paradigm for financial forecasting. CNNs can learn from large datasets and identify non-linear relationships, making them ideal for capturing the dynamic and multifaceted nature of financial markets. The goal of this study is to develop a robust CNN model that can predict market value with higher accuracy than traditional methods, thereby providing investors and analysts with a powerful tool for decision-making.

This research also aims to contribute to the growing body of literature on the application of deep learning in finance. By focusing on the Polish stock exchange, this study provides insights into an emerging market that has received less attention in the context of machine learning applications compared to more developed markets. The findings of this research could have implications for both academic research and practical applications, highlighting the potential of CNNs to revolutionize financial forecasting and investment strategies.

1.2 Theoretical Background

The capital structure of a company has been a central topic in modern finance theory for over four decades, reflecting its crucial role in corporate financial management and market dynamics [1]. The complexity and versatility of capital make the analysis of a company's capital structure essential for understanding its financial health and strategic direction. Capital, at its core, is an economic value with the potential for growth, leading to the creation of 'added value' such as profit or interest. This abstract notion of capital transforms into tangible financial resources or goods used in the production process, aiming to stimulate further development [2].

The conscious shaping and assessment of a company's capital structure involve analyzing the relationships and proportions between different types of financial resources, including equity and debt. The capital structure primarily refers to the configuration of a company's sources of financing, where equity represents ownership interest and debt constitutes borrowed funds subject to interest payments. Effective management of capital structure is vital as it influences a company's ability to generate added value, maintain financial stability, and pursue long-term growth [3].

Equity capital is pivotal in the functioning of a company, providing the necessary financial foundation for initiating business operations and safeguarding against potential losses. It also serves as a source of funding for ongoing business activities. Although equity financing is considered the safest option due to its non-repayment obligation, it often involves higher costs compared to debt financing. This higher cost is attributed to the opportunity cost of equity holders' funds and the potential dilution of ownership [4].

Conversely, debt capital arises from obligations to creditors and is characterized by lower acquisition costs. Debt financing can be beneficial due to tax advantages, as interest payments are tax-deductible, reducing the overall tax burden on the company. However, increased reliance on debt elevates financial risk and may adversely impact the company's creditworthiness. The challenge for financial managers lies in balancing the benefits of debt, such as cost advantages and financial leverage, against the associated risks [5].

The determinants of a company's capital structure are multifaceted, encompassing both internal factors related to the firm's operations and external factors from the macroeconomic environment. Internal factors, or microeconomic factors, include company size, industry characteristics, cost of capital, financial liquidity, risk tolerance, market position, asset structure, profitability, dividend policy, and investment strategy. These factors directly influence managerial decisions regarding the optimal mix of equity and debt [6].

External factors, or macroeconomic factors, encompass the broader economic context in which the company operates. These include the level of economic development, inflation rates, interest rates, fiscal policies, legal regulations, efficiency of the banking sector, technological advancements, and the development of financial markets. These macroeconomic determinants shape the external financing environment, impacting a company's ability to access capital and influencing its capital structure decisions [6].

Both microeconomic and macroeconomic factors significantly affect the choice of capital structure and, consequently, the market value of the enterprise. By thoroughly analyzing and adapting to these determinants, companies can optimize their financial strategies and enhance their market value.

1.3 Related Work

The market value of an enterprise reflects the price at which it can be bought or sold under open and competitive market conditions, assuming rational behavior and adequate information on both sides of the transaction [7]. This definition underscores the importance of transparency and information availability in determining fair market value. Various financial indicators are crucial for assessing an enterprise's value. Traditional indicators, such as return on equity, sales growth, fixed asset investments, and changes in working capital, provide insights into a company's financial performance. Additionally, competitive advantages and strategic positioning play a significant role in valuation [8].

Maximizing enterprise value is a fundamental objective in contemporary economic approaches, particularly for public companies listed on stock exchanges. For these companies, market value is often equated with market capitalization, which is the product of the total number of shares and their current market price. This metric provides a clear and quantifiable measure of a company's value in the financial market, essential for investors and shareholders evaluating the company's investment potential [9].

Understanding market value involves comparing a company's share prices to benchmark indices representing similar enterprises within the same industry. This comparative analysis helps eliminate the influence of general market trends, allowing a focus on the individual performance and value of the enterprise [10].

Over the past few decades, the financial sector has experienced significant transformation due to advancements in information technology. The implementation of machine learning (ML) and deep learning (DL) algorithms has revolutionized data analysis and investment strategy optimization. These technologies enable the processing and interpretation of vast amounts of financial data, offering greater accuracy and speed in predicting market trends and making investment decisions [11].

Machine learning algorithms, including support vector machines (SVM), decision trees, and random forests, have found widespread applications in finance. These methods are used for various tasks, such as regression analysis, real estate price forecasting, credit card fraud detection, and stock price prediction. ML algorithms excel in identifying patterns within data and improving predictive accuracy by minimizing errors [12].

Deep learning, a subset of machine learning, has garnered special interest for its ability to handle complex financial problems. Deep learning models, particularly convolutional neural networks (CNNs), are inspired by the human brain's neural networks, where information is transmitted between neurons through synapses. Similarly, in artificial neural networks, data moves between nodes (analogous to neurons) via connections whose strengths (weights) are adjusted during the learning process [13].

Neural networks' ability to learn and adapt by adjusting their weights based on training data allows them to recognize patterns, predict outcomes, and make decisions with remarkable accuracy. This capability makes neural networks particularly suitable for financial forecasting, where they can uncover trends and insights not readily apparent through traditional methods. Neural networks' applications in finance include image recognition for fraud detection, natural language processing for sentiment analysis, and complex time-series forecasting for stock prices and market trends [13].

In practice, the iterative adjustment of weights in neural networks enables these models to evolve and improve their predictive capabilities over time. By learning from historical data, neural networks can adapt to changing market conditions and provide valuable insights for financial decision-making. This adaptability and learning ability distinguish neural networks from traditional statistical methods, offering a powerful tool for financial analysis and forecasting [13].

2 Problem Formulation and Proposed Solution

2.1 Proposed Approach

This approach involves several key steps, including data collection and preprocessing, feature extraction, model architecture design, training, and evaluation. The aim is to leverage CNNs to capture complex patterns in financial data that traditional models may miss.

2.1.1 Data Collection and Preprocessing

The data for this study was obtained from two primary sources:

- **Company Financial Records**: The dataset includes Excel files from 350 companies listed on the Polish stock exchange (GPW), covering the period from 1998 to 2022 on a quarterly basis. These files contain comprehensive information about the companies' financial statuses at the end of each period. For this study, all data directly related to capital and asset structure, such as equity, debt, short-term and long-term liabilities, industry sector, and number of shares during the period, were extracted. Additionally, indirectly related economic indicators that involve capital or assets were considered.
- **Historical Market Data from Stooq.pl**: This source provides historical market data. By matching the dates and companies, the average market value of each company's stock on the balance sheet date was aligned.

The data preprocessing steps include:

- **Data Cleaning**: Handling missing values, outliers, and inconsistencies in the dataset to ensure the quality of data.
- **Feature Engineering**: Creating new features that may better capture the underlying financial dynamics. This includes adding columns that show the difference between quarters, two quarters, and four quarters to better represent the essence of the study. Additionally, ratios like debt-to-equity and return on assets were created. One-hot encoding was also employed for categorical variables to convert them into a format suitable for machine learning algorithms.
- **Normalization**: Using techniques such as `Standard Scaler` from `scikit-learn` to standardize the features, ensuring that they have a mean of zero and a standard deviation of one.
- **Dataset Preparation**: Two datasets were prepared for the experiments. The first dataset includes only the specific values directly related to the study. The second dataset encompasses all related factors, including indirectly related economic indicators, where capital and asset structure play a role. Additionally, two corresponding Data Frames were created to facilitate the experiments.

2.1.2 Model Architecture Design

The Convolutional Neural Network (CNN) model used in this study is designed to capture and learn patterns from the financial data. The architecture consists of several layers, each serving a specific purpose in the learning process. The model is implemented as follows:

```
class ModelCNN(nn.Module):
    def __init__(self, input_shape):
        super(ModelCNN, self).__init__()
        self.conv1 = nn.Conv2d(1, 16, kernel_size=(3, 1))
        self.conv2 = nn.Conv2d(16, 32, kernel_size=(3, 1))
        self.fc1 = nn.Linear(32 * (input_shape[2] - 4), 50)
        self.fc2 = nn.Linear(50, 1)

    def forward(self, x):
        x = torch.relu(self.conv1(x))
        x = torch.relu(self.conv2(x))
        x = x.view(x.size(0), -1)
        x = torch.relu(self.fc1(x))
        x = self.fc2(x)
        return x
```

- **Convolutional Layer 1 (conv1)**: This layer applies 16 convolution filters to the input data. The kernel size is (3, 1), meaning that the convolution operation slides a window of size 3 over the input data vertically while maintaining the full width of the input. This layer helps in detecting low-level features such as edges and textures.

- **Convolutional Layer 2 (conv2)**: This layer applies 32 convolution filters with the same kernel size of (3, 1). By increasing the number of filters, this layer can capture more complex patterns and features from the output of the first convolutional layer. The use of multiple convolutional layers allows the network to learn hierarchical feature representations.

- **Fully Connected Layer 1 (fc1)**: After flattening the output from the second convolutional layer, this layer consists of 50 neurons. The role of this fully connected layer is to interpret the high-level features extracted by the convolutional layers and learn the nonlinear combinations of these features. The number of input features to this layer is calculated as $32 \times (\text{input\_shape}[2] - 4)$, considering the dimensions of the flattened feature map.

- **Fully Connected Layer 2 (fc2)**: This is the output layer of the network with a single neuron. It takes the 50-dimensional input from the previous fully connected layer and outputs a single value, which is the predicted market value. This layer performs a linear transformation on the input features to generate the final prediction.

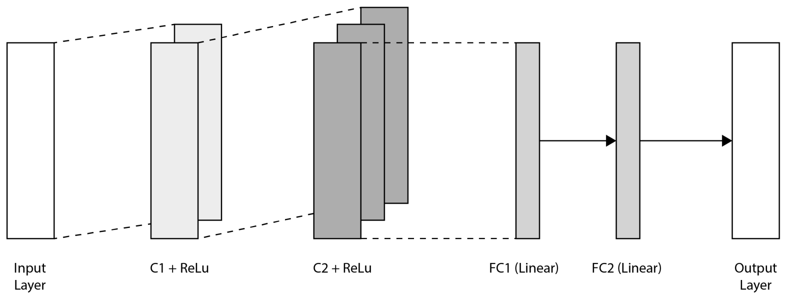

Figure 1: Convolutional Neural Network - Model Overview

- **Activation Functions**: The Rectified Linear Unit (ReLU) activation function is applied after each convolutional and fully connected layer, except for the output layer. The ReLU function introduces nonlinearity to the model, allowing it to learn complex patterns. ReLU is defined as $\mathrm{ReLU}(x) = \max(0, x)$, which helps in mitigating the vanishing gradient problem and accelerates convergence.
- **Flattening Operation**: Before passing the output of the convolutional layers to the fully connected layer, the feature maps are flattened using the `view` function. This operation reshapes the multidimensional tensor into a two-dimensional tensor, making it compatible with the fully connected layers.

2.1.3 Training and Evaluation

To train the CNN model, the dataset is split into training and testing sets using the `train_test_split` function from `scikit-learn`. This ensures that the model is evaluated on a separate subset of the data that it has not seen during training, providing a more accurate assessment of its performance. The training process involves the following key steps:

- **Optimization**: Optimizer Adam is utilized to minimize the loss function. That optimizer adjust the weights of the neural network to reduce the difference between the predicted and actual values.
- **Loss Function**: A suitable loss function like Mean Squared Error (MSE) is implemented to quantify the prediction errors. The choice of loss function is crucial as it guides the optimizer in updating the model parameters.
- **Evaluation Metrics**: The model performance is assessed using metrics such as Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and R-squared (`r2_score`). These metrics provide different perspectives on the accuracy and reliability of the predictions.

Mean Absolute Error (MAE): The Mean Absolute Error (MAE) measures the average absolute difference between the predicted and actual values. It provides a straightforward interpretation of the average error magnitude, making it an intuitive metric for assessing model performance. The formula for MAE is:

$$\mathrm{MAE} = \frac{1}{n}\sum_{i=1}^{n} |y_i - \hat{y}_i|$$

where y_i represents the actual value, $\hat{y}_i$ represents the predicted value, and n is the number of observations.

Root Mean Squared Error (RMSE): The Root Mean Squared Error (RMSE) is the square root of the Mean Squared Error (MSE) and provides an estimate of the standard deviation of the prediction errors. RMSE is more sensitive to large errors than MAE, making it a useful metric when large deviations are particularly undesirable. The formula for RMSE is:

$$\mathrm{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

where y_i represents the actual value, $\hat{y}_i$ represents the predicted value, and n is the number of observations.

R-squared ($\mathbf{R^2}$): The R-squared (R^2) metric represents the proportion of the variance in the dependent variable that is predictable from the independent variables. It provides a measure of how well the model captures the variability of the target variable. An R^2 value closer to 1 indicates a better fit. The formula for R^2 is:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

where y_i represents the actual value, $\hat{y}_i$ represents the predicted value, and $\bar{y}$ is the mean of the actual values.

To prevent overfitting, an early stopping mechanism is implemented. This technique halts the training process when the validation loss stops improving for a specified number of epochs (patience). By monitoring the validation loss and stopping early when there is no significant improvement, the model's generalization capability is maintained. The early stopping mechanism is integrated into the training loop as follows:

```
if test_loss < best_test_loss:
    best_test_loss = test_loss
    patience_counter = 0
else:
    patience_counter += 1
    if patience_counter >= patience:
        print("Early stopping triggered")
        break
```

By following this training and evaluation approach, we aim to develop a robust CNN model capable of accurately predicting market value based on changes in capital and asset structure.

2.2 Libraries

In this section, the libraries used in this study to implement the proposed solution for predicting market value using convolutional neural networks (CNNs) are described. Each library plays a crucial role in data preprocessing, model building, training, and evaluation.

- **pandas** - `pandas` is a powerful and flexible data analysis and manipulation library for Python. It provides data structures like DataFrame, which is ideal for handling tabular data. In this study, pandas is used for loading, cleaning, and organizing the financial dataset from the Polish stock exchange (GPW). The library's capabilities in handling large datasets efficiently make it essential for preprocessing tasks before feeding the data into the CNN model.
- **scikit-learn** - is a robust machine learning library in Python. It includes various tools for data preprocessing, model selection, and evaluation. In this study, several sublibraries are used:
  - `StandardScaler`: Used for standardizing features by removing the mean and scaling to unit variance. It ensures that all input features contribute equally to the model.
  - `train_test_split`: Splits the dataset into training and testing sets, facilitating model evaluation and ensuring that the model's performance is assessed on unseen data.
  - `mean_squared_error`, `mean_absolute_error`, `r2_score`: Metrics used to evaluate the performance of the CNN model, quantifying prediction errors and model accuracy.
- **PyTorch** - is an open-source machine learning library widely used for deep learning applications. It provides a flexible platform for building and training neural networks. In this study, several PyTorch sublibraries and modules are utilized:

  - `torch`: The core PyTorch library, which includes the tensor data structure and various mathematical operations essential for building neural networks.
  - `torch.nn`: Contains classes and functions to build neural network layers, used to define the architecture of the CNN.
  - `torch.optim`: Provides optimization algorithms such as SGD and Adam, used to adjust the weights of the neural network to minimize the loss function.
  - `torch.utils.data.DataLoader`, `TensorDataset`, `Dataset`: Facilitate loading and preprocessing data in a format suitable for training the neural network, handling batching, shuffling, and parallel loading of data for efficient training.

- **NumPy** - is a fundamental package for scientific computing in Python. It provides support for large multidimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays. In this study, NumPy is used for numerical operations and data manipulation, particularly when converting data between pandas Data Frames and PyTorch tensors.
- **Matplotlib** - is a plotting library for Python that provides tools for creating static, animated, and interactive visualizations. In this study, matplotlib is used to visualize the performance of the CNN model, such as plotting the training and validation loss over epochs. These visualizations are crucial for understanding the model's behavior and diagnosing any potential issues during training.

2.3 Algorithm

In this section, the essential functions and classes utilized in the implementation of the CNN model for predicting market value are presented. Each component's purpose and functionality are detailed to provide a comprehensive understanding of the model's architecture and operation.

StockDataset Class: This class is a custom dataset class that inherits from `torch.utils.data.Dataset`. It is designed to handle our financial data for use in PyTorch DataLoader. It initializes the dataset by converting the input features and targets into PyTorch tensors with float32 data type, provides the number of samples in the dataset, and retrieves the input features and target value at the specified index.

```
class StockDataset(Dataset):
    def __init__(self, X, y):
        self.X = torch.tensor(X, dtype=torch.float32)
        self.y = torch.tensor(y, dtype=torch.float32)

    def __len__(self):
        return len(self.X)

    def __getitem__(self, idx):
        return self.X[idx], self.y[idx]
```

ModelCNN Class: This class defines the architecture of our Convolutional Neural Network (CNN). It initializes the CNN layers, including convolutional layers and fully connected layers, and defines the forward pass of the network, applying ReLU activation after each layer and flattening the output before passing it to the fully connected layers.

```
class ModelCNN(nn.Module):
    def __init__(self, input_shape):
        super(ModelCNN, self).__init__()
        self.conv1 = nn.Conv2d(1, 16, kernel_size=(3, 1))
        self.conv2 = nn.Conv2d(16, 32, kernel_size=(3, 1))
        self.fc1 = nn.Linear(32 * (input_shape[2] - 4), 50)
        self.fc2 = nn.Linear(50, 1)

    def forward(self, x):
        x = torch.relu(self.conv1(x))
        x = torch.relu(self.conv2(x))
        x = x.view(x.size(0), -1)
        x = torch.relu(self.fc1(x))
        x = self.fc2(x)
        return x
```

prepare_data Function: This function extracts the input features and target values from the DataFrame. It drops the target column from the DataFrame to get the input features and separates the target values.

```
def prepare_data(df, target_column='target'):
    X = df.drop(columns=[target_column]).values
    y = df[target_column].values
    return X, y
```

reshape_data Function: This function reshapes the input features and target values for the CNN and splits the data into training and testing sets. It reshapes the input data to fit the CNN input requirements.

```
def reshape_data(X, y):
    X_train, X_test, y_train, y_test = train_test_split(X, y,
        test_size=0.2, random_state=42)
    X_train = X_train.reshape(-1, 1, X_train.shape[1], 1)
    X_test = X_test.reshape(-1, 1, X_test.shape[1], 1)
    y_train = y_train.reshape(-1, 1)
    y_test = y_test.reshape(-1, 1)
    return X_train, X_test, y_train, y_test
```

train_model Function: This function trains the CNN model using the provided data loaders, loss function, and optimizer. It performs the training loop, updating the model weights, and evaluates the model on the test data after each epoch, storing the training and testing losses.

```
def train_model(model, train_loader, test_loader, criterion,
    optimizer, epochs=50, patience=5):
    train_losses = []
    test_losses = []
    best_test_loss = float('inf')
    patience_counter = 0

    for epoch in range(epochs):
        model.train()
        running_loss = 0.0
        for inputs, targets in train_loader:
            optimizer.zero_grad()
            outputs = model(inputs).squeeze()
            loss = criterion(outputs, targets.squeeze())
            loss.backward()
            optimizer.step()
            running_loss += loss.item() * inputs.size(0)
        epoch_loss = running_loss / len(train_loader.dataset)
        train_losses.append(epoch_loss)

        model.eval()
        test_loss = 0.0
        with torch.no_grad():
            for inputs, targets in test_loader:
                outputs = model(inputs).squeeze()
                loss = criterion(outputs, targets.squeeze())
                test_loss += loss.item() * inputs.size(0)
        test_loss = test_loss / len(test_loader.dataset)
        test_losses.append(test_loss)

        print(f'Epoch {epoch + 1}/{epochs},
            Train Loss: {epoch_loss:.4f}, Test Loss: {test_loss:.4f}')

        #Early stopping mechanism
        if test_loss < best_test_loss:
            best_test_loss = test_loss
            patience_counter = 0
        else:
            patience_counter += 1
            if patience_counter >= patience:
                print("Early stopping triggered")
                break

    loss_df = pd.DataFrame({'Epoch': list(range(1, len(train_losses) + 1)),
        'Train Loss': train_losses[:len(test_losses)], 'Test Loss': test_losses})
    loss_df.to_csv("cnn_training_results.csv", index=False)
```

evaluate_model Function: This function evaluates the CNN model on the test data and returns the predictions and actual values. It sets the model to evaluation mode, makes predictions on the test data, and returns the results.

```
def evaluate_model(model, test_loader):
    model.eval()
    predictions, actuals = [], []
    with torch.no_grad():
        for inputs, targets in test_loader:
            outputs = model(inputs).squeeze()
            predictions.extend(outputs.cpu().numpy())
            actuals.extend(targets.cpu().numpy())
    return np.array(predictions), np.array(actuals)
```

These functions and classes form the core of our CNN implementation for predicting market value, handling data preparation, model architecture, training, and evaluation.

3 Experimental Results

3.1 Results and Analysis

Results of experiments on two datasets: `nn_df` and `mini_df`.

The `nn_df` dataset includes all columns directly and indirectly related to capital and asset structure, in addition to feature engineering techniques applied to enhance the dataset. This comprehensive dataset aims to capture a broader range of financial indicators and their relationships.

On the other hand, the `mini_df` dataset focuses exclusively on the direct state of capital and asset structure, with feature engineering applied as well. This dataset aims to streamline the input features to those most directly relevant to the model's objective.

The performance of the CNN model is evaluated using three metrics: Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), and R-squared (R^2). The results are summarized in Table 1.

Dataset	**RMSE**	**MAE**	R^2
`nn_df`	0.3757	0.0816	0.8632
`mini_df`	0.2554	0.0827	0.9286

Table 1: Performance of the CNN model on the `nn_df` and `mini_df` datasets.

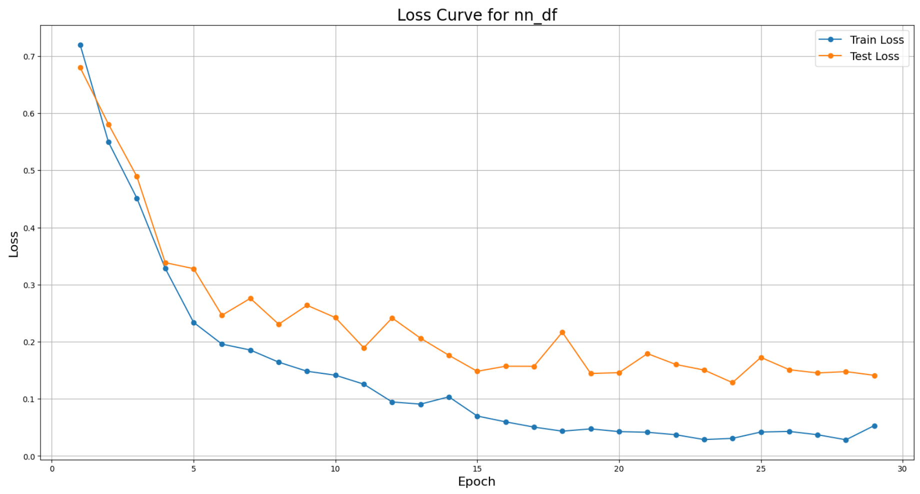

Figure 2: Loss Curve for Neural Network Model

nn_df Dataset:

- The training loss decreases steadily over the epochs, indicating that the model is learning and fitting the training data well.

- The test loss also decreases but shows more fluctuations compared to the training loss. This indicates that while the model is improving on the training data, it experiences some variability when evaluated on the test data.

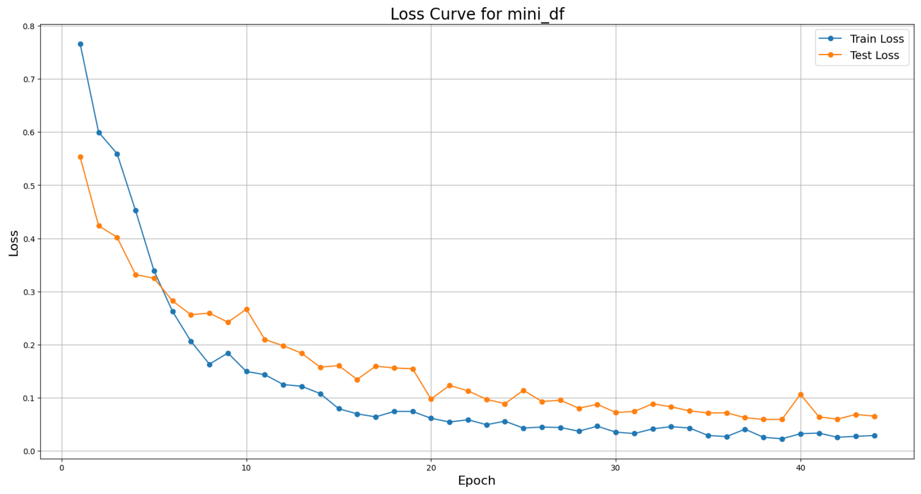

Figure 3: Loss Curve for Mini CNN Model

mini_df Dataset:

- Similar to the nn_df dataset, the training loss decreases consistently, showing effective learning by the model.
- The test loss decreases and shows fewer fluctuations compared to the nn_df dataset, suggesting a more stable performance on the test data.
- The test loss stabilizes around the 30th epoch, with minor fluctuations, indicating that the model generalizes well without significant overfitting.

From the results, we observe the following:

- The model achieved a lower RMSE on the `mini_df` dataset compared to the `nn_df` dataset. Specifically, the RMSE decreased from 0.3757 to 0.2554, indicating that the model's predictions are closer to the actual values on the `mini_df` dataset.
- The MAE is slightly higher on the `mini_df` dataset (0.0827) compared to the `nn_df` dataset (0.0816). Although the difference is small, it suggests that the average absolute error of the model's predictions is slightly greater on the `mini_df` dataset.

- The R^2 value, which indicates the proportion of variance in the dependent variable that is predictable from the independent variables, is higher for the `mini_df` dataset (0.9286) than for the `nn_df` dataset (0.8632). This suggests that the model explains a greater portion of the variance in the target variable for the `mini_df` dataset.

Overall, the `mini_df` dataset yielded better performance metrics, indicating that the CNN model may generalize better on this dataset. These results suggest that the model's ability to predict market values is more accurate and reliable when using the `mini_df` dataset.

3.2 Summary

The use of a Convolutional Neural Network (CNN) for predicting market value based on changes in capital and asset structure has proven to be effective. The experimental results demonstrate that the CNN model achieved a high performance, particularly on the `mini_df` dataset, where it reached an R^2 value of 92.86%. This indicates that the model was able to explain a substantial portion of the variance in the target variable, making its predictions highly reliable.

The results from the `nn_df` dataset, with an R^2 value of 86.32%, also underscore the model's capability to handle a broader range of financial indicators and relationships. However, the higher performance on the `mini_df` dataset suggests that focusing on direct capital and asset structure features may enhance the model's accuracy.

Despite the satisfactory performance, it is important to note that such a model might not always be sufficient in every context. Financial markets are influenced by a myriad of factors, some of which may not be fully captured by the datasets used. The complexity and dynamic nature of financial data mean that the model's performance can vary depending on the quality and relevance of the input data.

Moreover, while an R^2 value of 92.86% is impressive, there are scenarios where even higher accuracy may be required, particularly in high-stakes financial decision-making environments. Hence, further improvements and refinements to the model, as well as the inclusion of additional relevant features, could be necessary to achieve even better results.

In summary, the CNN model demonstrated a robust ability to predict market values with a high degree of accuracy. These results are promising and indicate that with continued development and adaptation, such models can become valuable tools in financial analysis and forecasting.

3.3 Limitations and Future Work

The model's performance is highly dependent on the quality and completeness of the financial data. Any inaccuracies or missing data can significantly affect the model's predictions. Additionally, the dataset used in this study is limited to companies listed on the Polish stock exchange (GPW). Extending the dataset to include companies from different regions and sectors could provide more generalized insights. The inclusion of more diverse data could help in understanding and predicting market values in a broader context.

While both datasets included several relevant features, there may be other important variables not considered in this study. Incorporating additional features, such as macroeconomic indicators, market sentiment, and geopolitical factors, could potentially improve the model's predictive accuracy. In particular, market sentiment is a major factor influencing market value. Investor sentiment, driven by news, social media, and overall market trends, can have significant impacts on stock prices and market movements. This study did not implement sentiment analysis, which is a critical component in understanding market behavior and making informed investment decisions. Integrating sentiment analysis into the model could enhance its ability to capture short-term market fluctuations and improve prediction accuracy.

To better capture the temporal dependencies in financial data, future work could explore the integration of temporal models like Recurrent Neural Networks (RNNs) or Long Short-Term Memory (LSTM) networks with CNNs. This hybrid approach could leverage the strengths of both architectures, improving the model's ability to predict market values over time. Testing other algorithms, such as LSTM, could provide additional insights and enhance the predictive performance by capturing sequential dependencies and long-term trends in the data.

Future research could focus on identifying and incorporating additional features that capture the underlying economic and market conditions. Feature selection techniques, such as Principal Component Analysis (PCA) or Recursive Feature Elimination (RFE), could be employed to identify the most relevant features. These techniques can help in reducing the dimensionality of the data, improving model efficiency, and potentially increasing predictive accuracy by focusing on the most impactful variables.

Implementing advanced regularization techniques, such as dropout or L2 regularization, can help prevent overfitting and improve the model's generalization capabilities. Dropout implementation can reduce the risk of overfitting by randomly omitting neurons during the training process, thus making the model more robust. Additionally, exploring model ensemble methods, where multiple models are combined to produce a single prediction, can further enhance performance and reliability. Model ensembles can capture a wider range of patterns and reduce the variance in predictions, leading to more stable and accurate outcomes.

4 Citations

[1] Saugata Banerjee, Almas Heshmati, Clas Wihlborg. The Dynamics of Capital Structure. (p. 1)
[2] Sylwia Betkowska. Determinanty struktury kapitału w przedsiębiorstwie. (p. 386)
[3] Sylwia Betkowska. Determinanty struktury kapitału w przedsiębiorstwie. Borowiecki, Czaja, Jaki 1998, s. 21 – cytowanie wtórne. (p. 387)
[4] A. Cwynar, W. Cwynar. Optymalizacja struktury kapitału i kalkulacja kosztu kapitału spółki, w: Finansowanie rozwoju przedsiębiorstwa, red. M. Panfil, Difin, Warszawa 2008, s. 57.
[5] A. Kwiecień. Struktura kapitału a stopa zwrotu – analiza przypadku. Studia Ekonomiczne 222 (2015), s. 139–152.
[6] Agnieszka Kurczewska. Determinanty struktury kapitałowej przedsiębiorstwa. Uniwersytet Łódzki. (pp. 327, 329)
[7] Sara Rupacz, Izabela Jonek-Kowalska. Model szacowania wartości przedsiębiorstwa na podstawie danych publikowanych przez Giełdę Papierów Wartościowych w Warszawie. (pp. 111-113)
[8] Adriana Kaszuba-Perz, Paweł Perz. Rola przedsiębiorcy w budowaniu wartości przedsiębiorstwa rodzinnego. (p. 135)
[9] Rappaport A. (1999), Wartość dla akcjonariuszy, Wig-Press, Warszawa.
[10] Damodaran A. (2017), Finanse korporacyjne. Teoria i praktyka, Onepress, Gliwice.
[11] Jesus Cuauhtemoc Tellez Gaytan, Karamath Ateeq, Aqila Rafiuddin, Haitham M. Alzoubi, Taher M. Ghazal, Tariq Ahamed Ahanger, Sunita Chaudhary, G. K. Viju. AI-Based Prediction of Capital Structure: Performance Comparison of ANN, SVM and LR Models.
[12] Sheng Chen, Hongxiang He. Stock Prediction Using Convolutional Neural Network. (pp. 3-6)
[13] Kevin Gurney. An Introduction to Neural Networks. University of Sheffield, pp. 3-6.
[14] Vijay Choubey. Understanding Recurrent Neural Network (RNN) and Long Short Term Memory (LSTM). Medium.