Zeppelin Universität

Department of Business & Economics

Chair for Empirical Finance and Econometrics

Prof. Dr. Franziska Peter

Bachelor thesis

Taking Color into the Blackbox

Training Convolutional Neural Networks with Red and Green Pixels of Intraday Return Data to Forecast Realized Volatility

Author:	Nicolas Bühringer
Student ID:	20204264
Program:	Corporate Management & Economics
Semester:	Spring 2024
Supervisor:	Prof. Dr. Franziska Peter
Submission date:	10.06.2024

Abstract

This thesis explores the use of convolutional neural networks (CNNs) to better forecast day-ahead Realized Volatility for 28 Dow Jones Industrial Average stocks using high-frequency return data. To reduce computational costs, this study transforms monthly minute-by-minute log returns into small 21x380 pixel images utilizing only their red and green color channels. These images train CNNs for volatility prediction. Results show that while the CNNs can forecast volatility, they did not consistently outperform the baseline model. The research underscores the need for further optimization of CNN architectures and suggests future integration of recurrent elements to enhance temporal pattern recognition.

Contents

List of Figures

List of Tables

1. Introduction

“In everyday language, volatility refers to the fluctuations observed in some phenomenon over time. Within economics, it is used slightly more formally to describe, without a specific implied metric, the variability of the random (unforeseen) component of a time series” (Andersen et al., 2006, p. 780).

Forecasting volatility is critical for a variety of reasons, primarily due to its significant implications for risk management, asset pricing, and portfolio optimization. Accurate volatility forecasts enable financial institutions to quantify the risk associated with their asset holdings more precisely. Moreover, volatility is a fundamental input in the pricing models of derivative instruments, such as options, where the expected future volatility of the underlying asset directly influences the option's premium. For investors and portfolio managers, understanding and anticipating volatility patterns helps in optimizing asset allocation to enhance returns while controlling for risk. Consequently, reliable volatility forecasting models are indispensable tools in the financial industry, contributing to informed decision-making and the efficient functioning of financial markets. (Poon & Granger, 2003)

Traditional approaches to modeling and forecasting volatility were pioneered by Engle (1982) with the introduction of the Autoregressive Conditional Heteroscedasticity (ARCH) model characterizes volatility as an evolving function of current information. The Generalized version of this model (GARCH) introduced by Bollerslev (1986) served as an industry standard for many years. In recent years, deep learning techniques have proved to be incredibly powerful across various domains while at the same time not relying on too many strict assumptions as classical statistical methods. The fields of language translation, speech recognition and computer vision were revolutionized by breakthroughs in deep learning methods. (Prince, 2024)

It comes with no surprise that deep learning approaches are gaining traction in financial prediction problems. Heaton et al. (2016) surveyed possible applications, concluding that “deep learning can detect and exploit interactions in the data that are, at least currently, invisible to any existing financial economic theory.” (p. 1). Nevertheless, they also underscore some drawbacks of deep learning methods, such as limited theoretical justification and suboptimal out-of-sample performance. A well-known issue is the “blackbox” nature of these models, meaning that while many deep learning methods outperform their benchmarks, the precise mechanisms underlying their performance remain unknown. (Prince, 2024)

A specific type of deep learning model, known as a convolutional neural network has revolutionized the field of image processing and classification. In these models, so called convolutional layers extract information from the pixel data of images to build their predictions. Employing convolutional neural networks for the use in financial problems is an emerging area of research. Sezer & Ozbayoglu (2020) explored the use of convolutional neural networks on 30-day bar charts of Dow 30 stocks to forecast algorithmic trading strategies. More recently, Reisenhofer et al. (2022) attempted to design a volatility forecasting convolutional network inspired by a traditional statistical model, the Heterogeneous Autoregressive model developed by Corsi (2009).

A notable drawback of all deep learning methods is their high computational cost. To address this issue, this thesis attempts a novel approach of constructing small-sized yet highly informational images out of high-frequency stock return data. This is achieved by utilizing the red and green color channels of pixels to directly represent minutely observed log returns in compact 21x380 pixel images. These images are then used to train convolutional neural networks to forecast the day-ahead Realized Volatility of 28 stocks of the Dow Jones Industrial Index.

The remainder of this paper is structured as follows: Section 2 introduces Realized Volatility as an estimator for actual volatility. Section 3 provides an overview of digital images and pixels. Section 4 discusses deep neural networks as universal function approximators. Section 5 introduces convolutional neural networks and their applications in image analysis. Section 6 outlines the methodology for creating the images and training the network. Section 7 presents the out-of-sample results. Section 8 discusses the findings and highlights areas for future research.

2. Realized Volatility

Volatility as the unobservable component that influences the variability of an asset's return over time is latent and thus can't be measured directly. Instead, a proxy variable needs to be defined which closely models the actual volatility. The ideal metric to measure volatility would be an integral over time as shown by Andersen et al. (2006). However, with the rising availability of high-frequency return data, new avenues to estimating actual volatility open up by utilizing Realized Volatility as an ex-post realization of the actual volatility. (Poon & Granger, 2003)

The Realized Volatility of day t is defined as the square root of the sum of squared intraday log returns:

$$RV_t = \sqrt{\sum_{i=1}^{M} r_{t,i}^2} \tag{1}$$

With the sampling frequency M as the number of intraday price observations and $r_{t,i} = log(P_{t,i}) - log(P_{t,i-1})$ as the log returns of the stock price P.

Realized Volatility has been shown to be an unbiased and, under certain frequency conditions for $M \longrightarrow \infty$, consistent estimator of actual volatility (Barndorff-Nielsen & Shephard, 2002). There are two advantages for using Realized Volatility as an estimation of actual volatility. First, the underlying volatility process can be measured without the need of an explicit model and second, the information in intraday data can be captured and utilized by standard time series methods (Andersen et al., 2006). This thesis leverages the availability of high-frequency return data for stocks of the Dow Jones Industrial Average to represent the observed minutely log returns as digital images.

3. The RGB Color Space and Digital Images

The RGB color space is an additive color model in which the three colors red, green and blue are combined to represent an image. It is the main system used in electronic devices such as computer screens or cameras. Digital images are composed of pixels with each pixel containing a value for the respective red, green and blue color channel. Computers operate using binary code, meaning all data is stored as either a 0 or a 1. Thus, each color channel in an (R, G, B) image is represented by a value between 0 and 255 as these numbers can be efficiently stored in 8 bits[1]. This allows for 16,777,216 possible colors[2]. For example, the pixel color (0, 0, 0) represents pure black, (255, 255, 255) represents pure white, and (255, 0, 0) represents pure red. An image is structured as a grid of pixels made up of rows and columns. A 20x40 pixel image[3] contains 20 rows and 40 columns of pixels resulting in 800 overall pixels. In contrast, a modern 24-megapixel camera captures images with 24 million pixels with dimensions of 4.000x6.000 pixels. These high-resolution images are too detailed and large for training convolutional neural networks, leading to memory overflow and unnecessarily long training times.

[1] $2^8 = 256$
[2] $256^3 =$ 16,777,216
[3] It is also common to define the shape of an array as (20, 40)

Some of the first highly accurate convolutional neural networks for image classification were typically trained on much smaller images such as 224x224 pixels (Simonyan & Zisserman, 2015). High camera resolutions are primarily necessary for printing but are impractical for network training purposes.

4. Neural Networks

This chapter provides a basic overview on the topic of deep learning and neural networks. A comprehensive summary can be found in the work of Goodfellow et at. (2016) and more recently by Prince (2024). A more theoretical approach is being provided by Roberts et al. (2021).

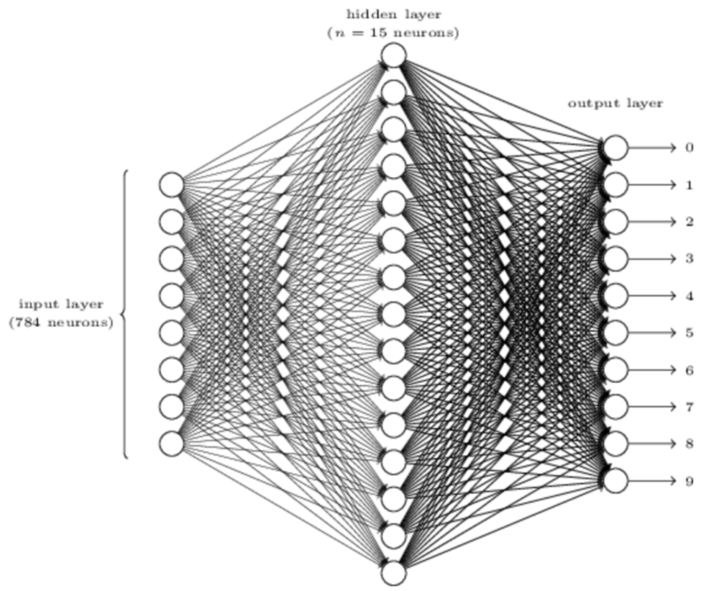
4.1. General Idea

The first ideas on neural networks were already published by McCulloch & Pitts (1943) and Rosenblatt (1958). The general idea was to mimic the learning of the human brain by introducing artificial neurons each capturing only a very small part of the provided data. These neurons are built up in multiple layers describing the depth of the network. The name *deep* learning stems from this. In theory, these networks can closely model any continuous function[4] (Hornik et al., 1989). The big breakthrough was the development of the backpropagation algorithm by Rumelhart et al. (1986) which is the method of how a neural network automatically improves itself during the training process. (Prince, 2024)

[4] Universal approximation theorem

To give a short summary and introduce some relevant terminology, the example of handwritten digit recognition is presented as studied by LeCun et al. (1998).

Figure 1: A Neural Network



Source: (Nielsen, 2015)

A neural network as seen in figure 1 can be thought of as layers of interconnected units called neurons. Each neuron tries to learn a small specific aspect of the given task and collectively, they form the neural network. In this figure, the layer on the left is called the input layer. It contains one neuron for each dimension of the input data. In this case, the handwritten digits are small greyscale images of 28x28 pixels, resulting in 784 input neurons. The layer on the right is called the output layer. It is made up of one neuron for the specific dimension of the task at hand. For regression tasks, the output layer contains a single neuron. In the case of classifying handwritten digits, the output layer consists of 10 neurons, corresponding to the 10 different digits. All layers in between are termed hidden layers since they are neither inputs nor outputs. Networks can include multiple of these hidden layers. (Nielsen, 2015)

As seen in figure 1, every neuron is connected to every other neuron in the adjacent layers. The value of one neuron in a hidden layer is computed as the weighted sum of all the neurons in the input layer plus a bias term. To introduce non-linearity, this weighted sum is passed through an activation function, such as the Rectified Linear Unit[5] function. Without it, the network would be a series of linear transformations,

[5] $ReLU(x) = max(0, x)$

making it equivalent to a single linear model regardless of how many layers it has. The optimal values of these weights and biases are parameters that are learned during the training of the neural network. The metric for how well these parameters are trained is determined by the value of the loss function which measures the size of the difference between the network's predictions and the actual labels of the training data. A common loss function used is the mean squared error, which the training process aims to minimize. The method of how the network tunes its weights and biases is an algorithm called backpropagation. To efficiently compute the necessary changes to reduce the loss function, the training data is divided into mini batches. Once all mini batches have been processed, an iteration called a training epoch is completed. The network is trained over multiple epochs until convergence is achieved, indicating that further training yields minimal improvement in performance. (Nielsen, 2015; Prince, 2024)

A fundamental concept in neural networks is the bias-variance trade-off which balances the model's ability to generalize to new data. High bias indicates that the model is too simple, leading to underfitting, where it fails to capture the underlying patterns in the training data. Conversely, high variance indicates that the model is too complex, leading to overfitting, where it performs extremely well on the training data but poorly on unseen data due to capturing noise and outliers. Effective neural network design seeks to find a sweet spot between bias and variance, ensuring the model is neither too simple nor too complex, thus achieving optimal predictive performance on new data. (Prince, 2024)

4.2. Unsuitability for Images

Dense neural networks are generally not suitable to work on images due to a couple of reasons. The main[6] and most intuitive can be understood by looking at the neuron connections inside the network. In dense neural networks, every input neuron is connected to every output neuron. Thus, each pixel in an image corresponds to a neuron in the input layer of the network. Together with a couple of hidden layers this leads to a significant number of parameters that need to be trained in each epoch. For example, given a standard colored 224x224 pixel image used in image classification as introduced in Chapter 3, the input layer would already have 150,528[7] trainable parameters. And with just one hidden layer of 512 neurons, which is not a lot for the complex task at hand, this number rises to 77 million[8]. This problem can be solved by sharing trainable parameters across the image pixels with the use of convolutional neural networks. (Goodfellow et al., 2016; Prince, 2024)

5. Convolutional Neural Networks

Convolutional neural networks are a type of neural network which are designed for data in a grid-like format like images. In recent years they are the reason for big leaps in the realms of automated object detection, facial recognition or image segmentation. They get their name from a mathematical operation called a convolution. (Goodfellow et al., 2016)

[6] Two other reasons exceeding the scope are statistical relations of nearby pixels and image stability under geometric transformations

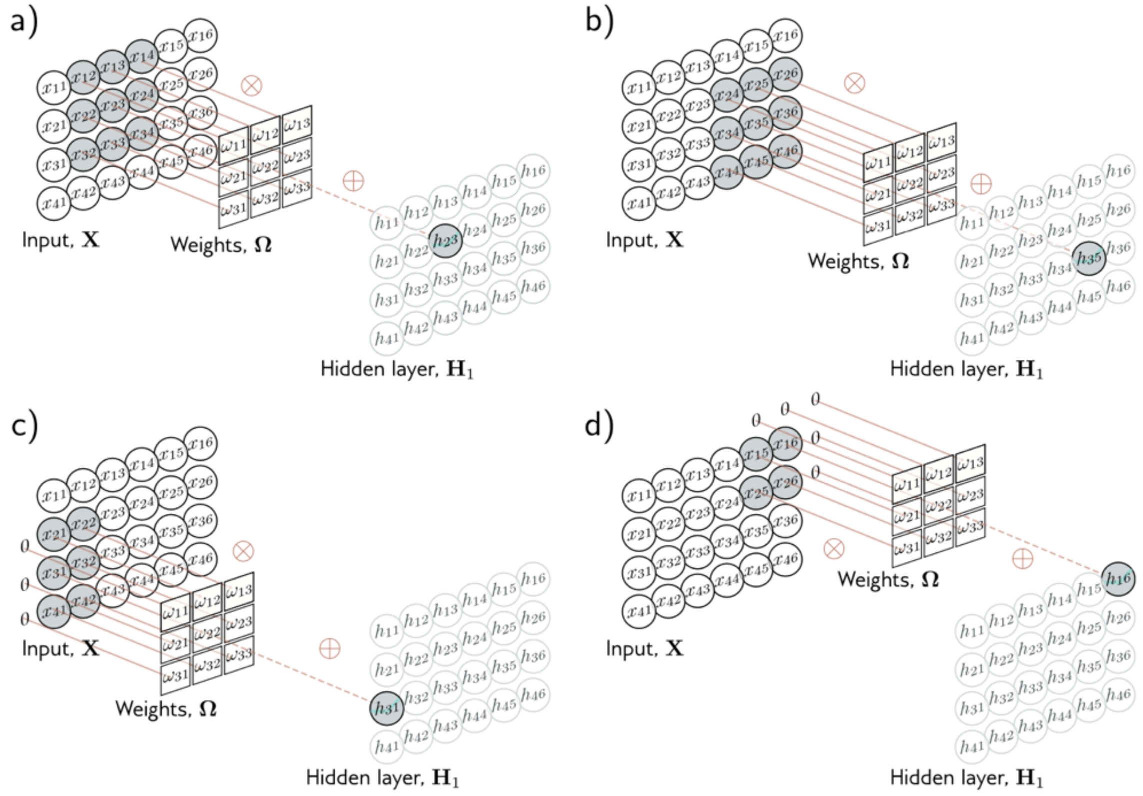
[7] $224 * 224 * 3 = 150{,}528$

[8] $150.528 * 512 = 77{,}070{,}336$

5.1. Convolutions

The main component of convolutional networks are their convolutional layers. A convolution is a mathematical operation and can be simplified to a kernel of weights sliding over subparts of the image to produce an output in the form of the weighted sum of nearby inputs as seen in figure 2. A kernel of size 3x3 could for example look like this $K = \begin{matrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{matrix}$. Each convolutional layer usually contains a set of kernels.

Figure 2: A convolutional layer



Source: (Prince, 2024)

One of the significant advantages of convolutional networks is that the optimal weights of the kernels do not need to be known beforehand. They are initialized randomly and then learned in the training process of the network.

Additionally, the number of trainable parameters in the entire network does not depend on the image size as it is the case with dense neural networks. Instead, in each convolutional layer, the number of parameters only depends on kernel size, the number of kernels and the number of input channels[9] plus one bias parameter for each kernel. The size of the image is irrelevant because, for each kernel, the parameters that need to be trained are shared among all pixels the kernel slides over during training. To give some further intuition on parameter sharing, if detecting for example edges using one special set of weights in a kernel is useful in one part of the image it should also be useful in other parts of it. (Prince, 2024)

5.2. Further Terminology

Some further terminology used in later parts of this thesis is explained.

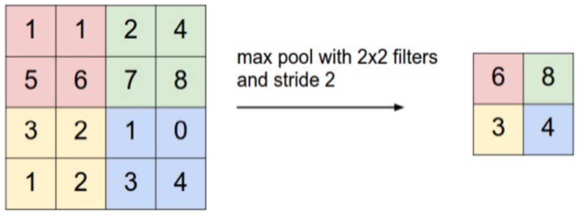
5.2.1. Padding

Padding ensures that the output of applying a kernel has the same size as the input. This is achieved by adding zeros to the edges of the input for the kernel to not start the sliding process at the very first pixel with actual information but rather at the added zero pixel while already *looking* at the first pixel with information. This can be seen in figure 2 c) and d). (Prince, 2024)

5.2.2. Pooling

Pooling is a downsampling method to further reduce the image dimensions used in training the network. The most common implementation is max pooling which takes the maximum value of a given non-overlapping subpart of the

Figure 3: Max Pooling



Source: (CS231n, 2024)

[9] In an RGB image the number of input channels is 3

input to produce an output. The size of the subpart is usually 2x2 but can be varied. Given a stride of 2, a 224x224x16 input will result in a 112x112x16 output. This is not only useful to avoid overfitting but also greatly reduces the number of parameters once the network is flattened to a dense representation as described in Chapter 4.2. (Goodfellow et al., 2016)

5.2.3. Dropout

A dropout is a form of regulating the network by randomly deactivating units given a predefined probability. Thus, their weights won't influence the output making the network less dependent on a couple of single weights. A side effect is that each epoch a slightly different version of the network is trained since a different set of units is deactivated. (Prince, 2024)

5.2.4. Early Stopping

Early stopping is one of the simplest methods to avoid overfitting the network during training. A small percentage of the training data is split and used as separate validation data during the training process. The network never sees these data points in the training process and therefore can't adjust its weights to better fit the validation data. If the loss function for the validation data does not decrease for a predefined set of epochs the training will be stopped even if the full number of training epochs have not been reached yet. Defining for how many epochs the validation loss is allowed to not decrease is the only hyperparameter of early stopping and sometimes called patience. It should not be chosen too small since the network could be caught in a local minimum before fully converging. (Prince, 2024)

6. Forecasting Methodology

The goal of this thesis is to forecast the day-ahead Realized Volatility of a given stock. For each stock in the dataset an ensemble of 5 convolutional neural networks was trained and evaluated. As a baseline a naïve forecast was used which takes the Realized Volatility value of the current day as a forecast for the following day.

6.1. Data

The dataset consists of high-frequency stock price data from 28 stocks of the Dow Jones Industrial Average index. The observation period reaches from January 4th, 2010 to December 31st, 2019 with minutely sampled values of the stock price from 9:35:00 am to 15:55:00 pm. Thus, the dataset of each stock consists of 2516 trading days each containing 381 stock price observations resulting in 958,596 rows. To calculate the Log Returns, the closing price of each minute was used. An overview on all stocks in the dataset can be found in Table 2 in the appendix

The dataset for each stock was split into a training set and a test set with the training set being made up of the training data itself and the validation data which is being used to monitor the training of the convolutional neural networks. The test set or evaluation data is used to compare the forecasts to the baseline model. A 85-15 split was used resulting in 2,138 observed days in the training set, and 378 observations in the test set. Since the model is dealing with time series data, the data split was not conducted via random sampling of the trading days but rather on its temporal axis. This means that based on their timestamp the oldest 85% trading days were used as training data, of which 20% were used as validation data. The last 15%, meaning the trading days closest to the present, were used as evaluation data in the test set. To calculate the Realized Volatility, equation (1) from chapter 2 was used on the closing price of each minute.

To evaluate the Realized Volatility forecasts of the convolutional neural networks the Mean Squared Error metric was used. It is defined as:

$$MSE\left(RV_{s,t}, \widehat{RV}_{s,t}\right) = \frac{1}{T}\sum_{t=1}^{T}\left(RV_{s,t} - \widehat{RV}_{s,t}\right)^2 \tag{2}$$

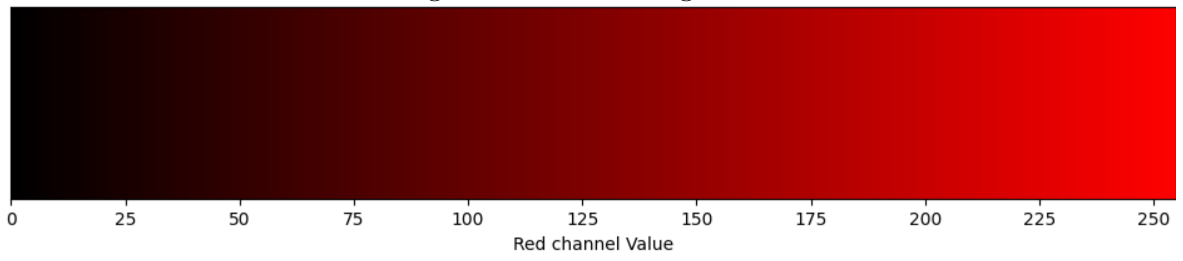
where $RV_{s,t}$ is the observed true Realized Volatility of stock s on day t and $\widehat{RV}_{s,t}$ is the Realized Volatility forecasted by the convolutional neural network. T is the number of days in the respective dataset.

6.2. Creating the Images

On average each month in a year consists of 21 trading days. The novel approach attempted in this thesis is to represent all the minutely Log Return data in a given 21-day period to forecast the day ahead Realized Volatility using only a single two-color channel 21x380 pixel image for the whole 21-day period. This is archived by the following steps:

First, 21-day subsequences are created on a rolling window basis for the whole dataset. This results in 2,117 data frames with 7,980 rows for 21 days of 380 return values in each day. For each subsequence the positive and negative log return values are split into two individual arrays. As an example, a positive log return at time t is being placed in the array for positive returns at index t while the same index in the array for negative returns is filled with a zero. This happens vice versa for negative log returns although the absolute value is used. Now both arrays are scaled using a Min-Max-Scaler which scales all the log returns into the interval [0, 255]. This means that the largest negative log return is being scaled to the value 255 while the smallest negative log return is being scaled to zero. The result are two arrays which can be used as color channels for an RGB-encoded image. Here, the scaled negative log returns were used as the red color channel and the scaled positive log returns were used as the green color channel. Hence, the intensity of the red or green pixel in the image, meaning its value from 0 to 255, is a direct representation of how large or small the log return was for the respective observation. As an example of how nuance these differences are, figure 4 shows a red color gradient starting at a value of zero and ending at its maximum value of 255 while both the green and blue color channels are kept at zero:

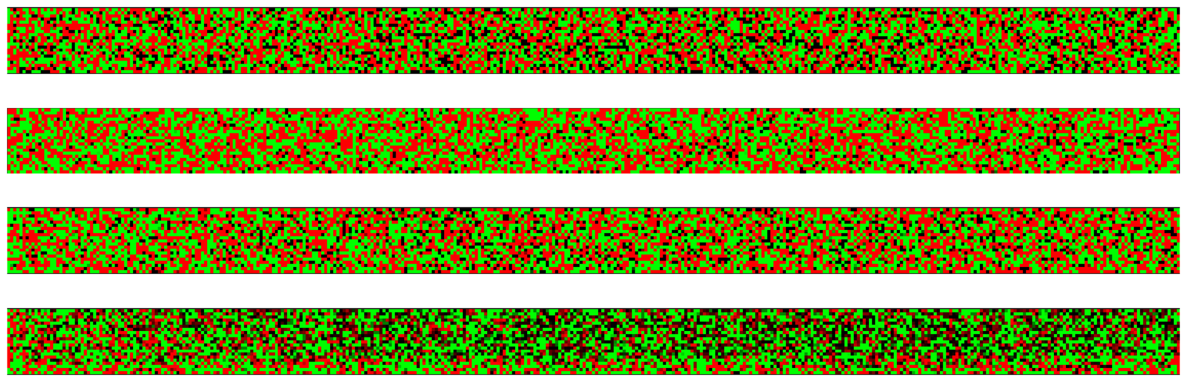
Figure 4: Red color gradient



To be able to correctly render the images a third array filled with zeros was added to represent the blue color channel. However, this additional channel was not used in the training of the convolutional networks since it doesn't contain any information.

Finally, the resulting (7980, 3) array is reshaped to an (21, 380, 3) array which represents the 3 color channels for 380 trading minutes in one of the 21-day observation windows. An example of four of these images can be seen in figure 5.

Figure 5: Images used to train the convolutional neural networks



Note: This figure shows 4 random images out of the dataset for stock Caterpillar. These 21x380x2 images contain the information of all 380 minutely intraday return values for 21 days. They are used in the training and testing of the convolutional neural networks.

These images contain two temporal axes with the x-axis representing observed minutes within a day (left to right) and the y-axis representing observed days (top to bottom). In total there are 1,693 images in the training set, 424 images in the validation set and 357 images in the evaluation set.

6.3. Training the Networks

For each stock an ensemble of 5 separate convolutional neural networks was trained using 100 training epochs on the same data to account for stochastic differences (Prince, 2024). As a loss function the mean squared error shown in equation 2 was chosen and as optimizer the Adam Algorithm (Kingma & Ba, 2014). Only the training data was used for training the networks while the validation data was used to monitor the training of the networks using an early stopping mechanism with a patience of 10 epochs. A small optimization of architecture was performed by testing different network designs on a single stock. A full list of tested parameters and their mean squared error on the Walmart data can be found in table 3 in the appendix.

6.3.1. Complex Model

Initially, a more complex model was trained successfully on the data of the Walmart stock. It was build using the following architecture: 3 blocks of convolutional layers followed by a max-pooling of (2, 2) and a dropout layer of 0.2. The first convolutional layer is made up of 16 kernels of size (3, 3) followed by 32 kernels of size (3, 3) and lastly 64 kernels of size (2, 2). Each convolutional layers uses the Rectified Linear Unit activation function and the sides of the images were padded with zeros while applying the kernels. After the last convolutional block, the output is flattened followed by 100 fully connected neurons also using the Rectified Linear Unit activation function and a final dropout layer of 0.7. The last element is a single prediction neuron using a linear activation function.

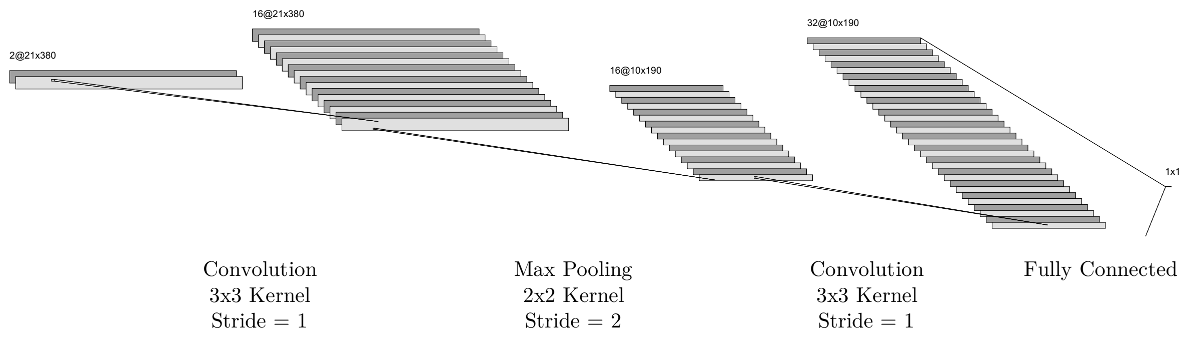
At first hand this model produced a mean squared error on the test set which was smaller than the baseline. However, after inspecting the predictions it turned out that for every image in the test set the network just predicted a value close to the mean Realized Volatility of the training set. In this case it turned out that the solution was to develop a far simpler model.

6.3.2. Simple Model

Some hyperparameter optimization of the final model used in this thesis was performed and the variations can be found in Table 3 in the appendix. A much simpler model was chosen using only two blocks of convolutional layers with 16 and 32 kernels. Both utilize the Rectified Linear Unit activation function and zero padding on the sides of the images. However, the first one uses kernels of size (3, 5) and the second one kernels of size (3, 3). The idea behind the rectangular kernel size was to allow for more detection of temporal intraday features which are depicted on the horizontal axis in the images. Only the first convolutional layer was followed by a (2, 2) max-pooling layer and a 0.2 dropout layer. Again, a very high dropout layer of 0.7 was used after the flattening operation with only one dense prediction neuron using the linear activation function.

The final model architecture is presented in figure 6. It consists of a total of 65,937 trainable, with 60,801 of these parameters generated during the flattening process preceding the fully connected prediction layer.

Figure 6: Final model architecture

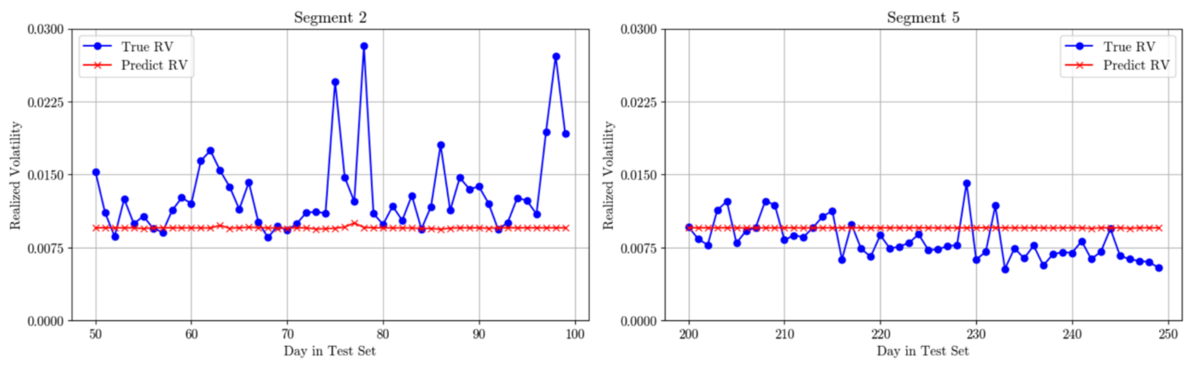


For each stock 5 individual networks were trained using the same training data to account for training stochasticity. The training was done for 100 epochs with a batch size of 16. However, early stopping usually halted the training for most networks at around 60 epochs. It turned out to be crucial to finetune the learning rate. The standard learning rate of 10^{-3} sometimes still produced a forecast of values very close to the Realized Volatility mean of the training set. Although 10^{-5} was also considered, setting the learning rate to 10^{-4} yielded the smallest mean squared errors on the test set.

7. Results

Although the initial complex model yielded a low mean squared error when evaluated on the test set, this low error was produced by the model only predicting a value close to the mean Realized Volatility of the training set. Figure 7 showcases this malfunction by plotting 100 observed Realized Volatility values of the test set of Walmart as well as the forecasted values of the convolutional network. Although the Mean Squared Error of the forecast produced by this convolutional network is quite low compared to the baseline, it is a forecast without any real-world value.

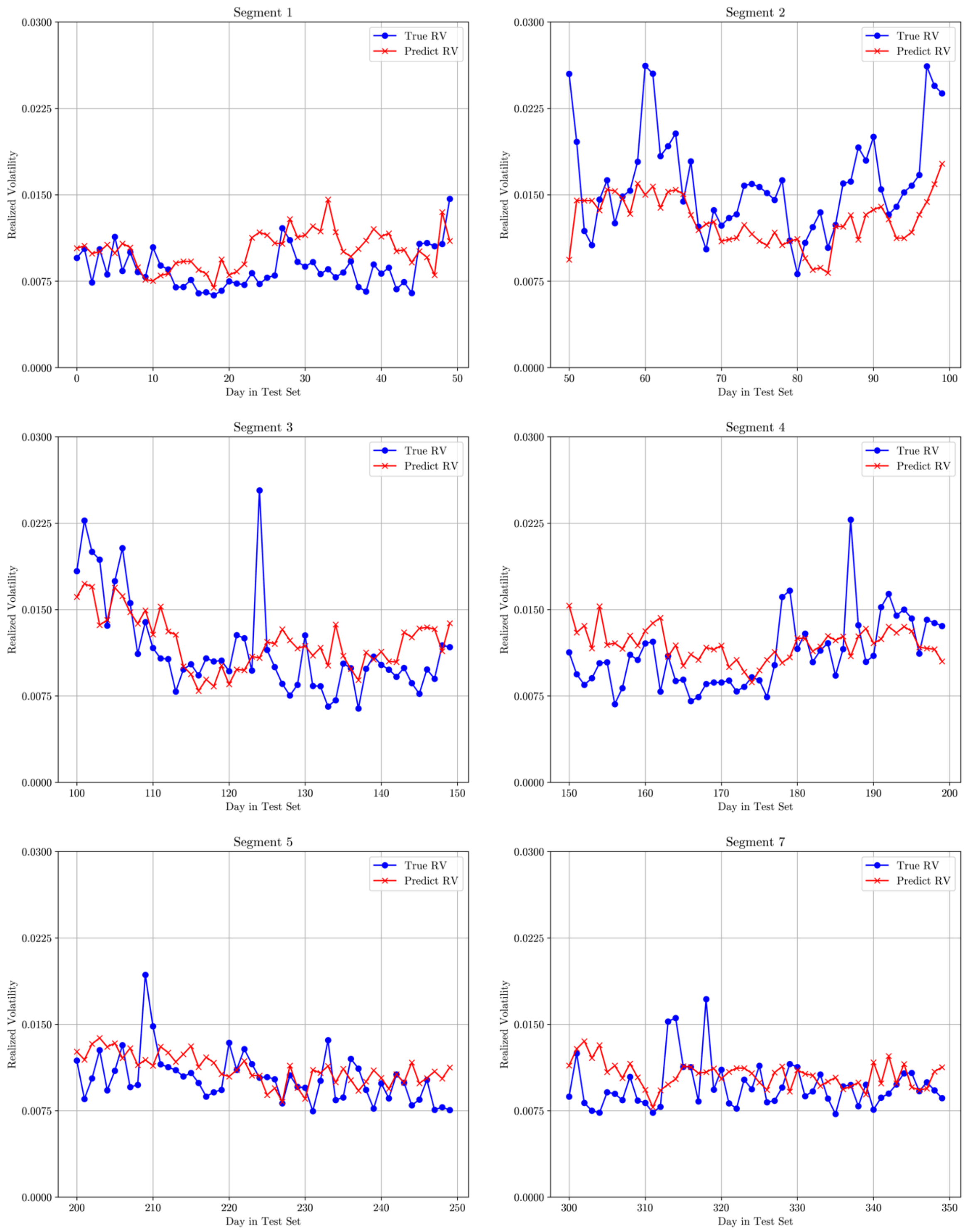
Figure 7: Network forecasting the Mean Realized Volatility



Note: This figure presents the observed realized volatility of Walmart stock along with 100 forecasted values generated by a single convolutional neural network. The two segments were selected from the 357 values in the test set for visualization purposes.

This error was fixed by adopting the simple model as described in 6.3.2. Henceforth, the model produced a more varying prediction which can be seen in figure 8. This figure illustrates the forecast of the top-performing convolutional network, identified as network 4 trained on the stock Amgen.

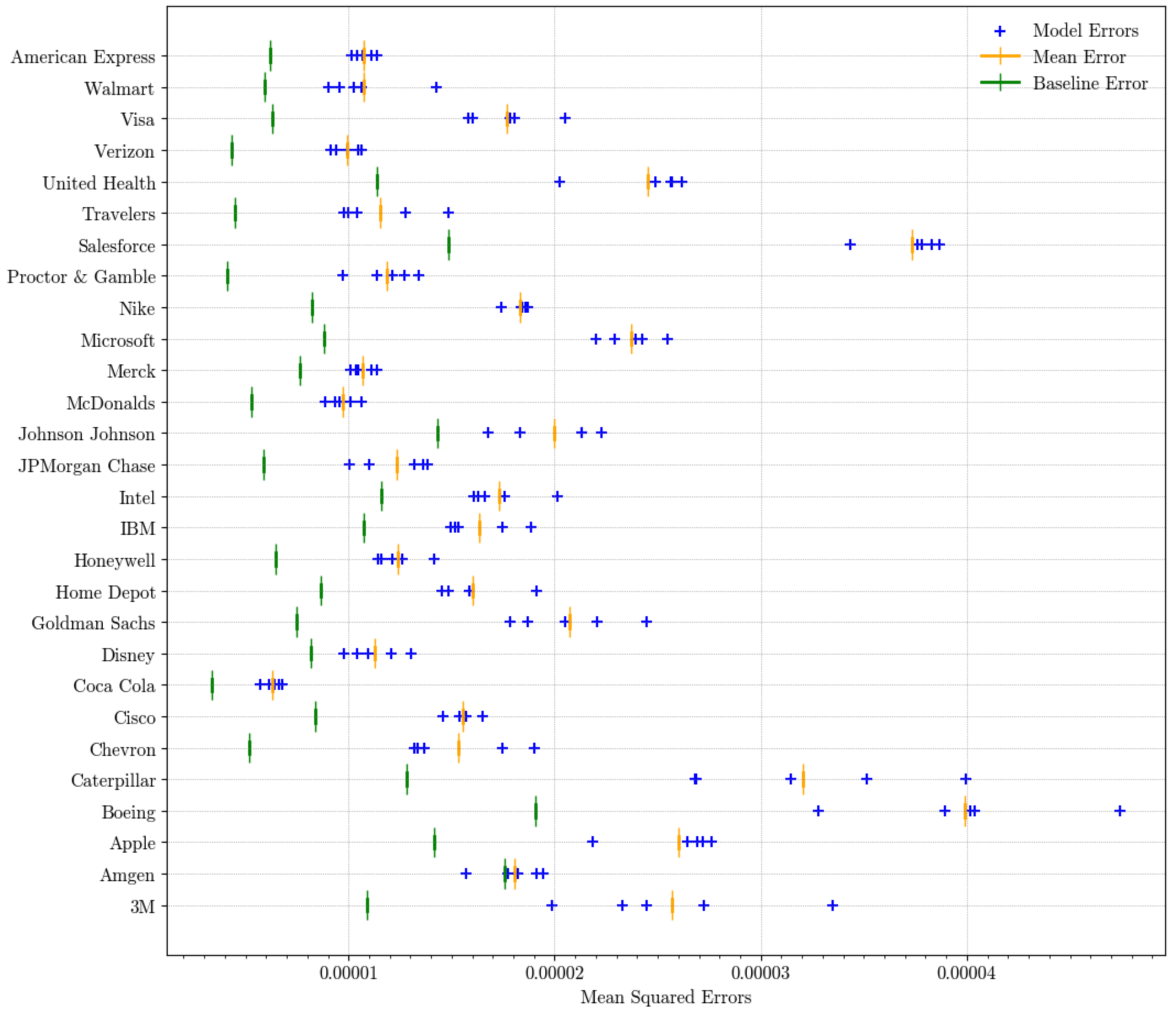
Figure 8: Amgen CNN 4 forecasting Realized Volatility



Note: This figure presents the observed Realized Volatility values for the stock Amgen as well as the forecasted values by convolutional neural network 4, which is the top-performing network of the whole dataset. The 50 days of segment 6 as well as the final 7 days of the test set were left out for visualization purposes.

The overall results can be seen in figure 9. The figure reports the Mean Squared Error for the 5 individually trained convolutional networks of each of the 28 stocks in blue with the mean of the 5 errors being highlighted by an orange vertical bar. The baseline error of each stock is marked by a green vertical bar.

Figure 9: Out-of-Sample Forecasting Performance



Note: This figure presents the mean squared error value for each convolutional neural network, alongside their respective means, represented in blue and orange. Additionally, the calculated baseline MSE for each stock is depicted in green.

This naïve baseline was calculated for each stock by simply forecasting the day-ahead Realized Volatility in the test set by taking the value of the previous day and calculating the Mean Squared Error. Considering the mean convolutional network error, this baseline error is beaten by none of the stocks.

Only a single convolutional network, CNN 4 of the stock Amgen, achieves a Mean Squared Error on the test set which is smaller than the baseline error for said stock. This model is marked bold table 1 which reports the numerical values of figure 9.

Table 1: Mean Squared Errors for 28 stocks

Stock	CNN 1	CNN 2	CNN 3	CNN 4	CNN 5	Mean	Baseline
3M	.0000335	.0000272	.0000233	.0000245	.0000199	.0000257	.0000109
American Express	.0000114	.0000101	.0000104	.0000107	.0000111	.0000107	.0000062
Amgen	.0000182	.0000191	.0000178	**.0000157**	.0000194	.0000180	.0000176
Apple	.0000269	.0000219	.0000272	.0000276	.0000264	.0000260	.0000142
Boeing	.0000402	.0000328	.0000389	.0000404	.0000474	.0000399	.0000191
Caterpillar	.0000268	.0000400	.0000351	.0000315	.0000268	.0000320	.0000128
Chevron	.0000133	.0000132	.0000175	.0000190	.0000136	.0000153	.0000052
Cisco	.0000156	.0000154	.0000165	.0000146	.0000157	.0000155	.0000084
Coca-Cola	.0000066	.0000068	.0000057	.0000061	.0000064	.0000063	.0000034
Disney	.0000098	.0000121	.0000104	.0000110	.0000131	.0000112	.0000082
Goldman Sachs	.0000187	.0000205	.0000220	.0000245	.0000178	.0000207	.0000075
Home Depot	.0000149	.0000145	.0000159	.0000158	.0000191	.0000160	.0000087
Honeywell	.0000114	.0000126	.0000116	.0000121	.0000141	.0000124	.0000065
IBM	.0000188	.0000150	.0000175	.0000153	.0000152	.0000164	.0000107
Intel	.0000166	.0000161	.0000201	.0000163	.0000176	.0000173	.0000116
JPMorgan Chase	.0000110	.0000138	.0000132	.0000100	.0000136	.0000123	.0000059
Johnson Johnson	.0000183	.0000223	.0000213	.0000167	.0000213	.0000200	.0000143
McDonalds	.0000101	.0000106	.0000095	.0000089	.0000094	.0000097	.0000053
Merck	.0000114	.0000101	.0000111	.0000104	.0000105	.0000107	.0000076
Microsoft	.0000220	.0000229	.0000239	.0000255	.0000242	.0000237	.0000088
Nike	.0000184	.0000184	.0000174	.0000187	.0000186	.0000183	.0000082
Proctor & Gamble	.0000097	.0000127	.0000121	.0000134	.0000114	.0000119	.0000041
Salesforce	.0000376	.0000387	.0000378	.0000383	.0000344	.0000374	.0000148
Travelers	.0000100	.0000149	.0000128	.0000098	.0000104	.0000116	.0000045
United Health	.0000202	.0000256	.0000249	.0000262	.0000257	.0000245	.0000114
Verizon	.0000106	.0000105	.0000099	.0000094	.0000091	.0000099	.0000043
Visa	.0000178	.0000160	.0000181	.0000205	.0000158	.0000177	.0000063
Walmart	.0000143	.0000102	.0000106	.0000095	.0000090	.0000107	.0000060

Note: This table displays the mean squared error of each convolutional network on the test set as well as the calculated baseline score

8. Discussion and Conclusion

This thesis investigates the approach of constructing compact images from high-frequency return data to input into a convolutional neural network for forecasting day-ahead Realized Volatility. The primary objective was to reduce computational costs during network training by encoding the log returns into red and green color channels. Consequently, each minute-level log return is represented through two color channels within a single pixel. The results presented in figure 9 and Table 1 indicate that the convolutional networks did not adequately capture sufficient information from the constructed images to reliably beat the calculated baseline score. Several potential explanations can account for these findings:

Firstly, the choice of architecture for the convolutional networks was arbitrary and only a minimal and non-systematic optimization of hyperparameters was performed. Also, different optimizers like Stochastic Gradient Descent or RMSprop could have been tested. Furthermore, the impact of the max pooling operation warrants closer examination. The constructed images of intra-day log returns contain an extremely high density of information with each pixel representing an observed value. The images contain two temporal dimensions, with the x-axis representing observed minutes within a day (left to right) and the y-axis representing observed days (top to bottom). This complexity may challenge the feature maps' ability to capture relevant information, and max pooling operations might distort these temporal dimensions. Future research could explore the integration of recurrent elements with convolutional layers to enhance the capture of temporal patterns, as employed by Gated Recurrent Unit models (Cho et al., 2014) and Long Short-Term Memory models (Hochreiter & Schmidhuber, 1997).

Secondly, convolutional neural networks may not be the most suitable model for forecasting Realized Volatility. Despite the flexibility of deep learning techniques in autonomously extracting necessary features from data, the results presented in this thesis underscore the blackbox-like nature of their internal mechanisms. The processes by which the convolutional network captures underlying information and generates forecasts remain unknown, thereby hindering any in-depth analysis for further insights. Consequently, statistical or economic interpretation of the produced forecasts is limited.

Thirdly, the choice of combining the data of one average trading month into an image was arbitrary too. Alternative approaches to data aggregation or input preparation could potentially improve the outcomes.

The limitations of this thesis lie mostly in its scope. As outlined, a more systematic approach to choosing a forecasting model as well as designing the model's architecture could have yielded better results. Furthermore, the baseline used in this thesis was calculated simply and a comparison with industry benchmarks, such as the GARCH or HAR models, would provide a more realistic benchmark. Additionally, the image construction approach relies on large, high-frequency datasets, which are not available for all types of assets.

In conclusion, while this thesis presents a novel approach to aggregating high-frequency return data into small sized images, the results indicate several areas for improvement. Future work should focus on systematic architectural and hyperparameter optimization, alternative data representation techniques, and the integration of recurrent elements to better capture temporal patterns. Furthermore, enhancing the interpretability of the model's predictions and conducting comparisons with established benchmarks can yield more reliable and comparable results.

Appendix

Table 2: Dataset overview

Stock	Observations	Mean RV
3M	2,516	.00873
American Express	2,516	.01009
Amgen	2,516	.01165
Apple	2,516	.01098
Boeing	2,516	.01119
Caterpillar	2,516	.01237
Chevron	2,516	.00998
Cisco	2,516	.01116
Coca-Cola	2,516	.00760
Disney	2,516	.00967
Goldman Sachs	2,516	.01213
Home Depot	2,516	.00982
Honeywell	2,516	.00940
IBM	2,516	.00861
Intel	2,516	.01196
JPMorgan Chase	2,516	.01132
Johnson Johnson	2,516	.00750
McDonalds	2,516	.00760
Merck	2,516	.00947
Microsoft	2,516	.01068
Nike	2,516	.01042
Proctor Gamble	2,516	.00755
Salesforce	2,516	.01552
Travelers	2,516	.00849
United Health	2,516	.01139
Verizon	2,516	.00886
Visa	2,516	.01041
Walmart	2,516	.00805

Note: This table reports the 28 stocks used in this thesis as well as their mean Realized Volatility.

Table 3: Network architecture optimization

Tested Version	Mean Squared Error	Forecasted the Mean
complex model, last drop 0.7, learning e-5	0.00005064	False
complex model, last drop 0.7, learning e-3	0.00004913	True
complex model, last drop 0.7	0.00004920	True
complex model, dense 100, last drop 0.7, learning e-5	0.00004900	False
complex model, dense 100, last drop 0.7, learning e-3	0.00004887	False
complex model	0.00005354	False
16 conv, 3-5 Kernel, pool 2-2, 1 Dense, learning e^4	0.00006327	False
16 conv, 3-5 Kernel, drop 0.7, 1 Dense, learning e-4	0.00004833	False
16 conv, 3-5 Kernel, drop 0.2, 1 Dense, learning e-4	0.00005003	False
16 conv, 3-5 Kernel, 1 Dense, learning e-4	0.00005588	False
16 conv, 3-10 Kernel, 1 Dense, learning e-4	0.00005340	False
16 conv, 2-15 Kernel, 1 Dense, learning e-4	0.00006318	False
16 conv 3-5, pool 2-2, drop 0.2, 32 conv 3-3, drop 0.7, 1 Dense, learning e-4	**0.00004739**	**False**
16 conv 3-5, drop 0.7, 32 conv 3-3, 1 Dense, learning e-4	0.00004999	False
16 conv 3-5, drop 0.2, 32 conv 3-3, drop 0.7, 1 Dense, learning e-4	0.00004842	False
16 conv 3-5, drop 0.2, 32 conv 3-3, 1 Dense, learning e-4	0.00004764	False

Note: The table lists the tested architectures and hyperparameters with the chosen model being written bold. The tests were performed on the Walmart data using a single network. The abbreviated layers are listed in sequential order. *16 conv 3-5* refers to a convolutional layer of 16 kernels with size 3x5.

Pool = Max Pooling, *drop* = dropout, *learning* = learning rate, *complex* = model in chapter 6.3.1

References

Andersen, T. G., Bollerslev, T., Christoffersen, P. F., & Diebold, F. X. (2006). VOLATILITY AND CORRELATION FORECASTING. *https://doi.org/10.1016/S1574-0706(05)01015-3.*

Barndorff-Nielsen, O. E., & Shephard, N. (2002). Econometric analysis of realized volatility and its use in estimating stochastic volatility models. In *J. R. Statist. Soc. B* (Vol. 64, Issue 2). https://academic.oup.com/jrsssb/article/64/2/253/7098420.

Bollerslev, T. (1986). Generalized autoregressive conditional heteroskedasticity. *Journal of Econometrics*, *31*(3), 307–327. https://doi.org/10.1016/0304-4076(86)90063-1.

Corsi, F. (2009). A simple approximate long-memory model of realized volatility. *Journal of Financial Econometrics*, *7*(2), 174–196. https://doi.org/10.1093/jjfinec/nbp001.

CS231n. (2024). CS231n Deep Learning for Computer Vision. *Stanford University.* https://cs231n.github.io/convolutional-networks/.

Engle, R. F. (1982). Autoregressive Conditional Heteroscedasticity with Estimates of the Variance of United Kingdom Inflation. *Econometrica*, *50*(4), 987. https://doi.org/10.2307/1912773.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press.

Heaton, J. B., Polson, N. G., & Witte, J. H. (2016). Deep Learning in Finance. *Digital Finance*, *5*(1), 1–2. https://doi.org/10.1007/s42521-023-00080-2.

Hornik, K., Stinchcombe, M., & White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, *2*(5), 359–366. https://doi.org/10.1016/0893-6080(89)90020-8.

Kingma, D. P., & Ba, J. (2014, December 22). Adam: A Method for Stochastic Optimization. *http://arxiv.org/abs/1412.6980.*

LeCun Y., Bottou L., Bengio Y., & Haffner P. (1998). Gradient-Based learning Applied to Document Recognition. *Proceedings of the IEEE*, *86*(11).

Mcculloch, W. S., & Pitts, W. (1943). A LOGICAL CALCULUS OF THE IDEAS IMMANENT IN NERVOUS ACTIVITY. *BULLETIN OF MATHEMATICAL BIOPHYSICS*, *5*.

Nielsen, M. A. (2015). Neural Networks and Deep Learning. *Determination Press.* http://neuralnetworksanddeeplearning.com.

Poon, S.-H., & Granger, C. W. J. (2003). Forecasting Volatility in Financial Markets: A Review. In *Journal of Economic Literature: Vol. XLI.*

Prince, S. J. D. (2024). Understanding Deep Learning. *http://udlbook.com.*

Reisenhofer, R., Bayer, X., & Hautsch, N. (2022). HARNet: A Convolutional Neural Network for Realized Volatility Forecasting. *arXiv preprint arXiv:2205.07719.*

Roberts, D. A., Yaida, S., & Hanin, B. (2021). The Principles of Deep Learning Theory. *https://doi.org/10.1017/9781009023405.*

Rosenblatt, F. (1958). THE PERCEPTRON: A PROBABILISTIC MODEL FOR INFORMATION STORAGE AND ORGANIZATION IN THE BRAIN. *Psychological Review*, *65*(6), 386–408.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, *323*(6088), 533–536. https://doi.org/10.1038/323533A0.

Sezer, O. B., & Ozbayoglu, A. M. (2020). Financial trading model with stock bar chart image time series with deep convolutional neural networks. *Intelligent Automation and Soft Computing*, *26*(2), 323–334. https://doi.org/10.31209/2018.100000065.

Simonyan, K., & Zisserman, A. (2015). VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION. *http://www.robots.ox.ac.uk/.*

Declaration of honor

I, Nicolas Bühringer, hereby declare on my honor that I have written this bachelor thesis with the title: “Taking Color into the Blackbox: Training Convolutional Neural Networks with Red and Green Pixels of Intraday Return Data to Forecast Realized Volatility” independently and without outside help.

The adoption of literal quotations, as well as the use of the thoughts of other authors I have marked at the appropriate places of the work. The thesis has not yet been submitted in the same or a similar form to any other German or foreign examination authority and has not yet been published. I am aware that a wrong explanation will have legal consequences.

Friedrichshafen, 10.06.2024 [illegible]

Place, Time Signature