Master thesis submitted in partial fulfillment of the requirements for the degree

Master of Science in Management and Technology

at Technische Universität München

Automated Architecture and Hyperparameter Optimization for Deep Neural Networks Applied in Forecasting the Cross-Section of Stock Returns

Reviewer:	Prof. Dr. Christoph Kaserer Department of Financial Management and Capital Markets TUM School of Management Technische Universität München
Advisor:	Dr. Matthias X. Hanauer, CFA
External Advisor:	Dr. Benjamin Moritz
Supervising Chair:	Chair of Financial Management and Capital Markets
School:	TUM School of Management
Study program:	Master in Management and Technology, (TUM-BWL)
Composed by:	Antonio Di Giovanni Matriculation number: 03743324 Tel: +39 338-5807237
Submitted on:	December 14th, 2022

Contents

List of Figures

List of Tables

List of Abbreviations

AutoML	Automated Machine Learning.
BNN	Benchmark Neural Network.
CNN	Convolutional Neural Network.
CPU	Central Processing Unit.
CRSP	Center for Research in Security Prices.
DFN	Deep Feedforward Neural Network.
GPU	Graphics Processing Unit.
HPO	Hyperparameter Optimization.
LSTM	Long-Short Term Memory.
MLP	Multilayer perceptron.
NAS	Neural Architecture Search.
NN	Neural Network.
NNI	Neural Network Intelligence.
OLS	Ordinary Least Squares.
ReLU	Rectified Linear Unit.
RNN	Recurrent Neural Network.

SGD Stochastic gradient descent.

SMBO Sequential Model-Based Optimization.

TPE Tree-Structured Parzen Estimator.

Chapter 1

Introduction

1.1 Forecasting stock returns with Neural Networks

One of the most crucial topics in financial research for both investors and researchers is the forecasting of stock returns. Recently, several Machine Learning models proposed in the literature have demonstrated the potential to outperform traditional linear prediction models. Following this trend, new, more advanced machine learning techniques such as deep learning are entering the stock returns forecasting literature, providing the potential for reaching even better performance levels. This has been made possible with the quick advancement of software and hardware in the Artificial Intelligence domain and with access to vast amounts of data.

This thesis may represent the natural next step in the field of stock returns forecasting through deep learning. Architecture design and hyperparameter tuning of deep learning models are crucial to their performance, even more than other, simpler, machine learning models comprised of a smaller set of tunable parameters. Advancements in what is called the AutoML - which stands for Automated Machine Learning - field are impressive and show a particular focus of part of the academic community.

AutoML aims at making Machine Learning models more accessible to a broader audience by transforming the process of training and tuning a Neural Network

(NN) in order to make it more structured and potentially more efficient. Given the fact that there is no single model that can be adapted to all tasks or datasets, as stated in Gridin (2022), re-adaptation of models, designing neural network architectures, and tuning of hyperparameters are just some of the tasks that AutoML can tackle could help the problem mentioned.

Classic prediction models have usually assumed the existence of a linear relationship between firm characteristics, or factors, and the expected returns of a stock. Nowadays, models such as Artificial Neural Networks can explain non-linear relationships and are also an excellent way to deal with the potential problems arising from the high number of factors that have been included in the literature during the past years (Harvey & Liu, 2019). Including all the documented factors would cause problems when using Ordinary Least Squares (OLS) or similar techniques due to multicollinearity between predictors when considering them on their own, and to the excessive dimensionality that exists when accounting for higher-order interactions between them. It is widely known that OLS cannot deal with a high number of dimensions as this approach would require.

Artificial Neural Networks are effective at predicting stock returns and explaining variations in the cross-section of stock returns. Some areas still need to be adequately explored, though, especially those regarding the design of the neural networks and the algorithmic research of optimal and robust hyperparameters.

The rest of the thesis is divided as follows: The next section contains an overview of the academic literature published in the different relevant areas of this research. Chapter 2 encapsulates the theoretical background constituting the baseline on which assumptions, analyses, and empirical methods are built. Specifically, the field of Empirical Asset Pricing is described, and its recent development is briefly analyzed. The theoretical background of machine learning, specifically of neural networks, is explored, as well as the central concept of AutoML and a description of an exemplary algorithm used in the field, the Tree-Structured Parzen Estimator (TPE), which will also be part of the empirical analysis. Chapter 3 contains the full description of data, methodology, and evaluation metrics used during the different types of experiments conducted, and finally, Chapter 4 contains a detailed overview of the results obtained and an interpretation on their meaning. Chapter 5 contains a summary of the thesis.

1.2 Literature Review

The application of Machine Learning techniques in the relevant areas is relatively recent and has been conducted under multiple fashions. Some examples are Moritz and Zimmermann (2016) who apply tree-based models to portfolio sorting, Kelly et al. (2019), who use dimensionality reduction methods to estimate factor pricing models. Hanauer and Kalsbach (2022) show that return forecasts performed by several machine learning models lead to superior out-of-sample long-short returns compared to traditional linear models, in the setting of emerging markets. Gu et al. (2020) perform an extensive comparison of machine learning models in predicting the cross-section of US-stock returns, including boosted regression trees, random forests, and several neural network models.

Messmer (2017) applies a Deep Feedforward Neural Network (DFN) to the cross-section of US-stock returns to assess the model's performance, and Feng et al. (2018) do the same by applying several neural network architectures to the same problem. L. Chen et al. (2019) move in a different direction and propose a non-linear asset pricing model determined through a deep neural network in order to estimate the stochastic discount factor that explains asset returns from the conditional moment constraints implied by no-arbitrage.

The benchmark models employed in asset pricing are often based on linear factor models deriving from Fama and French (1992, 1993) and Fama and French (2014) which will be used in the empirical analysis as part of the performance evaluation of the developed model. These factor models are, in turn, evolutions of the Capital Asset Pricing Model originally published in Sharpe (1964) and Lintner (1965).

The literature around architecture and hyperparameters optimization is evolving quickly, new algorithms are being developed, and the possibility of implementing them in multiple fields is now more concrete than ever. Bergstra and Bengio (2012) have originally proposed the random search method for optimizing neural networks hyperparameters. Given the high dimensionality of the problem, the algorithms have since evolved to several branches of possible algorithms, one of them are genetic algorithms, exemplified by Domashova et al. (2021), who focus on their application to the architecture and hyperparameters optimization problem. Bischl et al. (2021) provide an overview of current and state-of-the-art hyperparameter optimization methods and practical problems that could arise with their implementation, as well as some possible solutions for them. Some presented algorithms include Bayesian Optimization, Multifidelity, and Hyperband. Gridin (2022) contains valuable and practical information on the framework used to implement all the analyses present in this thesis, called Neural Network Intelligence[1], published from Microsoft as a Python library (more details in Section 2.4.2).

1.3 Main Findings

I design the search experiment with a broad search space in order to inject as little bias as possible and leave the burden of finding a good-performing network onto the optimization algorithm. Results are compared with the entire series of trials that were part of the experiment, with a linear model and with a benchmark neural network adapted from Gu et al. (2020). This section outlines the main findings of the thesis:

Hyperparameters optimization techniques are able to find robust sets of hyperparameters

The experiment routine set up in this study shows that the performance of the best network found is robust to small changes in one specific hyperparameter, which is considered among the most sensitive ones to change in neural networks, the learning rate. The results of this network are comparable with those of the benchmark neural network.

It is possible to forecast the cross-section of stock returns using automatically tuned Neural Networks

Over all the trials of the experiment, several of them result in bad performances (most of them worse than the linear benchmark), some of them fail numerically in performing predictions, and some others show good in-sample performance measures, which also translate into good out-of-sample performance. Specifically, 873 networks out of 1000 trials have either been stopped before completing their train-

[1] github.com/Microsoft/nni

ing procedure or numerically failed at predicting the cross-section of stock returns. The good-performing networks can overcome the linear model and show significant over-performance over a factor model, the Fama-French 5 Factors Model augmented with the Momentum and Short-Term reversal factors in this case. However, the top-performing network does not outperform the benchmark neural network.

Model Interpretation techniques for neural networks can lead to interesting insights and is comparable across manually and automatically designed networks

I use the Integrated Gradients technique to get insights into the importance of each firm characteristics. These results provide helpful insights into the neural network models and the kind of predictors that mainly contribute to the predicted returns. Given the high number of features, it is hard to provide conclusive interpretations on where the neural networks tend to gain insights from, but analyzing the most important predictors can provide a rough idea of the inner workings of the so-called black-box algorithm.

Chapter 2

Theoretical Framework

2.1 Empirical Asset Pricing

Asset pricing is the study of models explaining the prices and returns of various securities. Empirical Asset Pricing, specifically, can be defined as the connection point between asset pricing theory and asset prices. The field has historically been focused on testing asset pricing theories on asset prices and, at the same time, on establishing the characteristics (or factors) pertaining to individual assets that influence their prices and how this happens.

There is one canonical problem in asset pricing, which is measuring asset risk premia. Literature in this field is divided in two main strands: The first focuses on cross-sectional analyses and the second on analyses based on time-series data. Time-series analyses aim to explain changes in expected asset returns over time. Some examples are Welch and Goyal (2008) and Rapach and Zhou (2013). Cross-sectional analyses, on the other hand, focus on how expected returns vary across different assets (of the same asset class) as a function of asset-level characteristics. The aim is thus to understand what underlying characteristics of an asset influence its expected returns. Examples can be found in Fama and French (2007), Lewellen (2014), and Hanauer and Lauterbach (2018).

Research in asset pricing has historically been dominated by linear models, and partly still is. Running cross-sectional regressions of future stock returns on a few lagged firm characteristics is the standard approach to understanding the relations between these factors and the relative asset returns. The three- and five-factor models published in Fama and French (1993) and Fama and French (2015) have achieved great success in the financial industry; they relate cross-sectional returns to some factors, which are market, size and value, for the three-factor model, with the two additional factors being the profitability and investment ones for the five-factor model. These two models are still seen as the benchmark in most of the academic literature. Following their success, several new factors have been identified and published as potential predictors capable of explaining additional variation in cross-sectional expected returns. The existence of this large number of factors has repeatedly been attributed to a "publication bias" (A. Y. Chen & Zimmermann, 2019). There have been multiple experiments of collecting and performing a census of all the factors published in academia, Harvey and Liu (2019) show that over 400 factors have been published since 1963. A. Y. Chen and Zimmermann (2021) reproduce a large number of published cross-sectional stock return predictors, replicating 319 characteristics drawn from previous meta-studies (with different t-stats in certain cases).

The linear regression approach has many limitations, as pointed out in the previous chapter. The most important one is the fact that these models cannot handle the above-mentioned large number of predictors. Considering that often company characteristics are highly correlated with each other, traditional prediction methods may not work, or they should be applied to a small subset of the potentially available data. Machine learning naturally poses itself as a potential candidate for a step forward in empirical asset pricing research.

Non-linear models, and between them, several machine learning models, have helped in solving these problems. Algorithms like artificial neural networks have been applied as well, and they have been shown to possess the ability to overcome common downsides as they generally allow for more extensive sets of predictors, still work when predictors are correlated with each other, and allow for more flexible functional forms given the possibility to apply non-linearities.

In this work, I will focus on forecasting the cross-section of stock returns, trying to understand whether networks found and tuned through AutoML optimization algorithms can predict stock returns given the set of underlying firm characteristics at any given point in time.

2.2 Machine Learning

The definition of machine learning is not always unique, as it can be context-specific and encompasses a wide range of techniques and methodologies. According to Mitchell (1997), it can be described through the following definition: "A computer program is said to learn from experience E with respect to some class of task T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E." To allow these algorithms to learn from experience, there are several techniques applied, in the subset of machine learning called supervised learning, the models are usually fed examples to be processed, such that they can compare their result with the real one of the examples. Machine learning thus represents algorithms that aim to learn a function that performs a task. In order to achieve this, optimization techniques are applied to the algorithm in order to improve the value of a metric used to measure its performance.

Machine learning models are usually well fit in making predictions in high dimensional settings, this peculiarity enhances their flexibility relative to more traditional prediction techniques applied to econometrics. This increased flexibility, however, brings a higher propensity of overfitting the data during the learning phase, often called training. To avoid this, regularization techniques are applied in order to support the model in discovering a function underlying the data-generating process and not just a function that matches the empirical distribution of the specific subset of data fed to the algorithm for the training phase. These functions are often perfectly replicating the data distribution seen during training, resulting in them being likely not applicable to unseen or "out-of-sample" data. On the other hand, with the correct application of tools to avoid overfitting, machine learning models provide higher hopes of successfully approximating the likely complex data-generating process underlying asset risk premia.

2.3 Artificial Neural Networks

Artificial Neural Networks are a particular type of machine learning model built out of a connected set of simple units called neurons, divided into multiple layers called hidden layers. These are inspired by animal neural systems, especially the brain, which is built out of large webs of interconnected neurons transmitting signals between each other. Individual neurons take a set of real-valued inputs, calculate a linear combination of them, and then, depending on the task being solved (e.g., regression, classification, etc...), output one or more real-valued numbers that represent the prediction related to that input set. Figure 2.1 shows a neural network example.

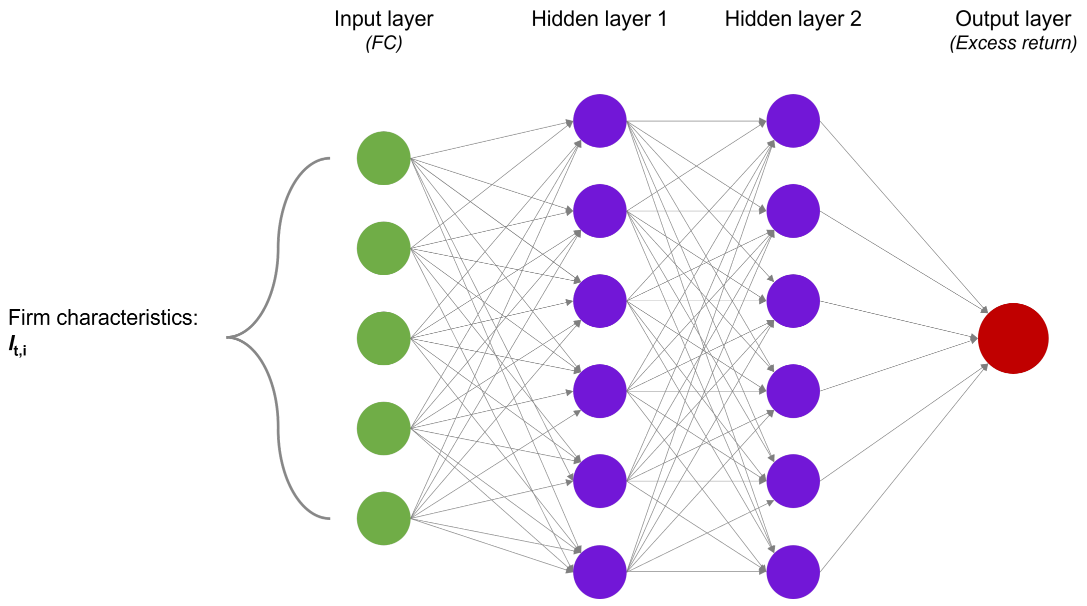


Figure 2.1: The picture shows an exemplary neural network and its macro components.

Artificial neural networks have a core characteristic, described by the universal approximation theorem (Cybenko, 1989; Hornik et al., 1989), which states that no matter the function $f(x)$ underlying a data-generating process, there exists a neural network with a sufficient number of hidden layers and neurons that can approximate that function. This theorem describes a highly flexible algorithm with the theoretical potential of learning about any data distribution and discovering the underlying relationships between predictors and targets.

Artificial neural networks can thus be considered among the most powerful machine learning models. At the same time, though, they are one of the least transparent and interpretable ones and among the models with the highest number of hyperparameters.

The parameters tuned via the training process are of two types: weights and biases. Most other parameters are included in the umbrella term hyperparameter and have to be set by the neural network designer (often called data scientist).

Apart from the already mentioned neurons, important elements of an artificial neural network are activation functions, objective functions, and the optimization algorithm.

2.3.1 Main components

Objective function

Also called cost function or loss function. It is used to evaluate a candidate solution for the neural network model parameters. The candidate solution consists of a specific set of weights and biases chosen for mapping the input data to one or more outputs. The loss function represents the "distance" between the predicted and real outputs, a measure of the error. This measure is used in the optimization process (backpropagation) to update network parameters in order to minimize the error produced by the network in the following tuning process steps. The loss function used in the empirical part of the thesis is the Mean Squared Error, defined as

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \tilde{y}_i\right)^2, \tag{2.1}$$

where y_i is the true target value, $\tilde{y}_i$ is the predicted target.

Optimization algorithm

As mentioned, machine learning models, undergo a process called training in order to approximate the function that maps features to target variables. This process is obtained by minimizing the cost function through an optimization algorithm.

Gradient descent is an iterative optimization algorithm commonly used in machine learning and especially with neural networks. It is applied with differentiable functions by calculating the gradient of the loss function with the current model parameters, scaling it with a parameter called learning rate, and using the result to update the current weights set.The process is written as

$$W' = W - \alpha \cdot \nabla_W f(W), \tag{2.2}$$

where W represents the set of weights of the neural network, $\nabla_W f(W)$ represents the gradients of the loss function computed with regards to all weights W, and

α is the learning rate, it scales by how much the model parameters are shifted towards the chosen direction (the same equation is applied to all biases b). Steps are made toward the negative gradient when minimizing a loss function. If the objective function had to be maximized, steps would be made in the opposite direction.

Stochastic gradient descent (SGD) is commonly used with large or deep neural networks. It calculates the gradient from a small random subset of the data as opposed to the conventional gradient descent, which uses the entire training set. This approximation gives up accuracy for a significant speedup of the optimization procedure.

The training process, and thus the learning performed by the network, is obtained through an algorithm called backpropagation, popularized by Rumelhart et al. (1986). This procedure entails the application of SGD to a complex function such as a neural network. It consists in dividing the training dataset into mini-batches and handling each one of them separately such that the algorithm goes through the training set several times. Each pass is called an epoch. The backpropagation algorithm follows five steps:

1. The input layer receives a mini-batch and sends it on to the first hidden layer.

2. The output of all neurons is computed by using weights and biases, the activation function is then applied to these values, and then the result is passed through to the following layer. The process is repeated until the last layer generates its output. This part is usually called the ”forward pass”.

3. The loss function is applied to the output to determine the neural network’s error following the forward pass.

4. The gradients of the cost function with respect to the parameters (weights and biases) are then calculated using the chain rule.

5. Weights and biases are updated following Equation 2.2 in order to minimize the loss function.

Activation function

The activation function is the non-linearity applied to data passing through a neuron before being fed to the next layer of the network. There are several choices for the nonlinear activation function, such as the Rectified Linear Unit (ReLU), Sigmoid, and others; these are often relatively simple functions applied multiple times by each neuron.

In this empirical analysis, potential choices of activation function are limited to ReLU and LeakyReLU. these two are defined as

$$\mathrm{ReLU}\left(x\right) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{otherwise} \end{cases} \tag{2.3}$$

and

$$\mathrm{LeakyReLU}\left(x\right) = \begin{cases} 0.01x & \text{if } x < 0 \\ x & \text{otherwise.} \end{cases} \tag{2.4}$$

Both activation functions are simple and do not require heavy computations. Compared to traditional activation functions such as sigmoid or the hyperbolic tangent, the main advantage they bring is the absence of problems such as the ”saturation” and ”vanishing gradients”. The first one occurs when the activation function is constrained in values, such as the sigmoid function, which can take values from -1 to 1, and thus outliers in the input distribution tend to have constant values; this causes the network to be only sensitive to changes in input values which are around the mean of the distribution, and insensitive to bigger and smaller inputs. The second problem, caused by the same principle of compressing large input values to a small range of values, occurs mainly when the network has several hidden layers. Gradients, essential for optimizing the neural networks, generally become small (e.g., smaller than 1). When these are multiplied several times due to the high number of hidden layers, gradients become too small for the training to work effectively (i.e., they ”vanish”).

ReLU, on the other hand, can suffer from a problem called ”dead ReLU”. This problem occurs when a neuron predicts only negative values, thus forwarding a value of 0 after ReLU is applied. It is difficult to change that situation during training, given that the derivative of ReLU at 0 is 0, and weights consequently do not change after an update step. In these cases, neurons are effectively not contributing to the neural network and are thus considered ”dead”. A potential solution for this is to apply LeakyReLU, which includes a slight slope in the negative range as seen in Equation 2.4.

2.3.2 Deep Learning

Increasingly complex tasks assigned to neural networks have posed the problem of having the correct representation of the data, which is fundamental for performance. For specific tasks, it is almost impossible to manually provide a formally accurate data representation, for example, in the case of speech or image recognition. Deep learning has joined the research scene as a potential solution for this problem, as this subset of artificial neural networks introduces representations built out of other, simpler representations. This type of algorithm allows the computer to learn not only the function to perform a task but also the correct representation of the data provided as input. These neural networks have been called deep networks to emphasize the contrast with shallow networks, which usually consist of 1 or 2 hidden layers. In comparison, most deep neural networks consist of 4 to 10 hidden layers, and the numbers are much higher in more advanced use cases.

Deep Learning models are thus a subset of artificial neural networks; they are becoming increasingly popular for tasks such as image classification, speech recognition, and robotics.

Deep Learning includes a comprehensive series of possible architectures and model structures, such as Convolutional Neural Network (CNN), Recurrent Neural Network (RNN), Long-Short Term Memory (LSTM), and others. The Deep Feedforward Neural Network (DFN), also known as Multilayer perceptron (MLP), is the workhorse model in this domain. Its basic implementation consists in an equation that maps a collection of input values to one or more output values as classic artificial neural networks do; the difference is in the fact that the number of layers in a DFN is higher than those in traditional artificial neural networks, as mentioned.

DFNs will be the only focus of the analyses carried out in the following chapters.

2.3.3 Regularization Techniques

When dealing with deep learning, regularization techniques are of the uttermost importance. They provide tools to avoid what is usually called overfitting. This phenomenon arises when the model learns to fit the training data perfectly, effectively mimicking it and not learning the true data-generating distribution underlying the task at hand. Examples of regularization methods applied in this study follow in this section. Goodfellow et al. (2016) is the general reference I used for describing the following methodologies, more information can be found in their book.

Dropout

The idea of dropout is quite simple; it consists in randomly dropping a subset of the neurons during the training phase of a neural network; in this way, the network is less complex, and the ability to fit the data is reduced, thus reducing potential overfitting as well. The deactivation of neurons happens with a predefined probability P, and it is applied at each forward pass and consecutive backpropagation step. The probability P is an additional hyperparameter that must be tuned.

An example of the difference between a normal neural network and one in which dropout is implemented can be seen in Figure 2.2; around one-third of the neurons are not active and thus not contributing to the neural network learning process,

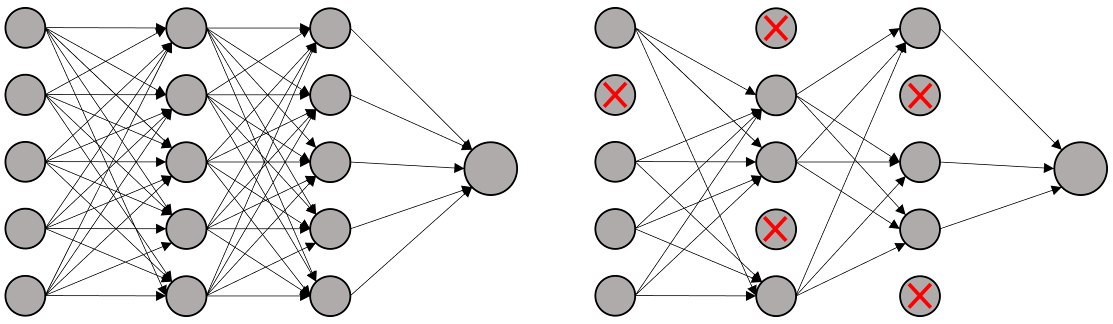


Figure 2.2: The figure shows an example of how the architecture of a neural network changes when dropout is implemented.

Early stopping

Deep learning networks are usually trained on a subset of the data, and during the training process, they are tested multiple times on another portion of data called validation set. Performance measures on the validation set should give an understanding of whether the network is generalizing or just overfitting the training data. The overarching idea of early stopping is thus to monitor the evolution of the cost function relative to the validation set and stop the training when the validation loss does not improve for a certain number of iterations. Stopping earlier than expected contributes to avoiding over-fitting, as shown by Goodfellow et al. (2016).

L1 and L2 regularization

L1 and L2 regularization are two examples of parameter norm penalty, both have commonly been used for decades, even before the advent of artificial neural networks. Logistic or linear regressions are just two examples of models that could benefit from such penalty terms. The implementation is relatively straightforward, a parameter norm penalty is added to the objective function. These norms may be considered constraints that the training optimization method must adhere to when optimizing the objective function (called loss function in the case of neural networks). In a generalized form, we could denote the regularized objective function as

$$\tilde{L}\left(w; X; y\right) = L\left(w; X; y\right) + \lambda\Omega\left(w\right), \tag{2.5}$$

where L is the loss function of the model, $\Omega(\theta)$ is the norm penalty, and $\lambda \in [0, \infty)$ is a hyperparameter that scales the contribution of the norm penalty term. A model is considered more "regularized" with larger λ and not regularized if λ equals 0.

The two methods owe their name to the L1 and L2 norm of a vector, in this case, the weights matrix.

L2 regularization, also known as weight decay, is the most common type of the two. It extends the loss function by a regularization term defined as

$$L2(W) = ||W||_2^2 = \sum_i \sum_j w_{ij}^2, \tag{2.6}$$

where W represents the weights matrix.

L1 regularization extends the loss function by the sum of the absolute values of the weights, it is defined as

$$L1(W) = ||w||_1 = \sum_i \sum_j |w_{ij}| \tag{2.7}$$

When these are combined, the resulting loss function can be written as

$$\tilde{L}\left(w; X; y\right) = L\left(w; X; y\right) + \lambda_1 L1(w) + \frac{\lambda_2}{2} L2(w) \tag{2.8}$$

The regularization terms are scaled by the parameters λ, which need to be tuned in order to improve the performance of the neural network.

An intuition from these two kinds of regularization is that both make weights slightly smaller and force their updates toward zero. Specifically, L1 encourages some weights to assume the value zero, performing a feature selection. At the same time, L2 pushes weights towards zero but without them actually assuming that value, thus providing less importance to individual neurons. Both achieve the objective of simplifying the model and making it less prone to overfitting the training data.

Batch Normalization

Batch normalization is a well-known technique first published in Ioffe and Szegedy (2015); in this publication, the authors try to tackle one major problem that occurs during the training of deep neural networks, the internal covariate shift, defined as the changes in the distribution of each layer's inputs during training, as the parameters of the previous layers change. This phenomenon has often been tackled by lowering learning rates and paying careful attention to initializing the weights of the neural network. Still, it has been shown that applying Batch Normalization helps in combating the problem and, at the same time, allows for much larger learning rates to be used and demands less attention on initializing weights. Its primary use is to control the variability of predictors across different network regions.

In order to reduce the internal covariate shift, this technique aims at fixing the distribution of the layer inputs as the training progresses. Specifically, the standard normal distribution is the one most often implemented in practice.

Batch Normalization, as defined on Goodfellow et al. (2016), replaces the minibatch of activations of a layer, named H, with $\gamma H' + \beta$, where H' is calculated as

$$H' = \frac{H - \mu}{\sigma}, \tag{2.9}$$

During training, μ and σ are defined as

$$\mu = \frac{1}{m} \sum_i H_{i,:}, \tag{2.10}$$

and

$$\sigma = \sqrt{\delta + \frac{1}{m} \sum_i (H - \mu)_i^2}, \tag{2.11}$$

where δ is a small positive value to avoid undefined gradients. γ and β are two hyperparameters that allow the standard normal distribution to be scaled to other distributions in order to avoid losing the expressive power of more complicated networks.

When testing on a test set, μ and σ are replaced with running averages collected during training time.

There is still debate on what the best implementation method for Batch normalization is(i.e., should it be applied before or after the activation function), as well as on what are its specific effects during training deep neural networks and what makes it so effective and powerful, with new publications updating the overarching hypotheses and confuting some previous ones. I will thus not touch upon this point in detail as it is not relevant to this research.

2.4 AutoML

2.4.1 General Description

AutoML is a recent topic in the machine learning research community which aims at creating methods and algorithms to automate the process of designing, tuning, and deploying machine learning models. These techniques allow non-data scientists to use machine learning techniques without needing experts in the field. Moreover, it would support data scientists when applying models to different datasets or tasks. As already stated, there is no "silver bullet model" to solve all problems with (Gridin, 2022). Each dataset and each application would need an architecture design and a hyperparameter tuning routine. These two processes could potentially be automated and made more efficient through AutoML.

AutoML can be divided into six main branches, each one of them comprising multiple tasks that the set of techniques can provide support in completing:

- Hyperparameter Optimization (HPO)
- Neural Architecture Search (NAS)
- Feature Engineering
- Model Compression
- Injection of new Deep Learning techniques into existing models
- Adjusting models to new datasets

As stated in the beginning, computing resources that are currently available to researchers make it possible for them to test multiple architectures and hyperparameter combinations to select an optimal one. Finding the optimal architecture and set of hyperparameters would be an illusion given the size of the search spaces and the non-convexity of the optimization problem (Goodfellow et al., 2015).

Automated hyperparameter tuning involves applying optimization algorithms to find an optimal set of hyperparameters for the model being trained, usually without setting previous assumptions regarding the final model architecture. Hyperparameters cannot be learned from data and are typically set manually by the neural network designer.

In order to perform hyperparameter tuning, there must thus be a routine one level above the classic neural network training one, which is focused on tuning hyperparameters as well as the network architecture. Several techniques, such as one-shot neural architecture search, would allow completing this task without large numbers of iterations but do not apply to this work.

Hyperparameter optimization is another technique, among the most prominent ones in the field. It can be considered as an experiment run in order to find an optimal, or good enough, set of hyperparameters to optimize the performance of a machine learning model for the given task on the given dataset. An HPO experiment should contain at least four elements:

- Search space
- Trial
- Tuning algorithm
- Performance measure

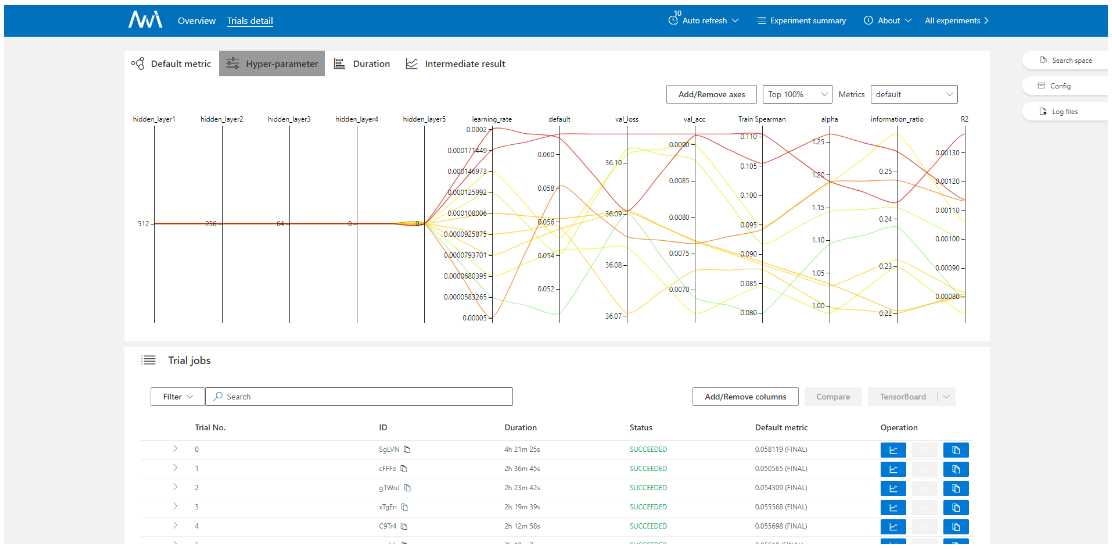
The trial is the single model being trained and tested with a set of hyperparameters drawn from the search space.These are usually chosen from a tuning algorithm, which can either be a simple grid search or random search, or one of the algorithms tailored for AutoML tasks, such as the Tree-Parzen Estimator, which will be described in section 2.4.3. The tuning algorithm performs the decision on which hyperparameter should be tested next in the experiment and, through the evaluation of the performance measure obtained at the end of each trial, tries to draw better-performing sets of hyperparameters as the trial progresses.

2.4.2 Microsoft's Neural Network Intelligence

Neural Network Intelligence (NNI) is an open-source library developed by Microsoft that allows to implement AutoML techniques in Python while using either one of the most popular neural network frameworks for the same programming language, such as PyTorch, Tensorflow, and others.

I will focus on hyperparameter optimization without considering the other use cases of the library. The workflow is divided into experiments, which enclose the entire optimization routine. A set of embedded algorithms for finding architectures and tuning hyperparameters is already included, as well as the runtime for the whole experiment. A dashboard to view the experiment details, as well as those regarding every single trial, is also part of the library. Figure 2.3 shows an exemplary view of this dashboard.

There are five main components in an NNI optimization experiment:



Trial No.	ID	Duration	Status	Default metric
0	SgLVN	4h 21m 25s	SUCCEEDED	0.058119 (FINAL)
1	cFFFe	2h 36m 45s	SUCCEEDED	0.050565 (FINAL)
2	g1WoJ	2h 23m 42s	SUCCEEDED	0.054309 (FINAL)
3	xTgEn	2h 19m 39s	SUCCEEDED	0.055568 (FINAL)
4	C9Tr4	2h 12m 58s	SUCCEEDED	0.055698 (FINAL)

Figure 2.3: The image shows an exemplary view of a grid search experiment on a single network in which the only changing parameter is the learning rate. Several performance measures can be included for completeness in addition to the "default" measure, which is used by the tuning algorithm to perform its optimization routine.

Trial

The trial is a script that trains and eventually tests the model with a specific set of parameters and returns a metric used to evaluate the model's performance compared to other models in the same experiment. It is effectively the training of a single neural network as it would be implemented without performing a hyperparameter optimization routine, with two added components: the hyperparameters have to be set by the tuning algorithm, and the results have to be fed back to the same algorithm in order to decide which neural network will be trained next.

Search space

The search space defines the set of all possible hyperparameters and architectures that are tunable for the optimized model, as well as the values they could take. The number of hyperparameters in the search space is finite, but in most cases, the possible combination of values that they create makes it impossible for the tuner to test all of them; that is why a tuning algorithm is necessary for the experiment. Search spaces can take both discrete and continuous values depending on the chosen tuning algorithm.

Tuner

The tuner is an optimization algorithm that selects the set of hyperparameters to be tested, starts the trial, receives the result, and determines the new set of hyperparameters for the subsequent trial. NNI provides several choices and allows the user to define a customized tuner. The Tree-Structured Parzen Estimator is one of the tuners originally implemented by the library and will be discussed in the following section.

Performance measures

The performance measure is simply a metric that is passed to the tuning algorithm during and at the end of each trial. This measure is used to optimize the model performance during the experiment. It usually is either the validation loss value, which would be minimized or an accuracy value, which would be maximized. The performance measure used by the tuner is just one, called ”default” in NNI, but the user can show more performance measures simultaneously in the dashboard with the proper setup.

Partial performance measures (sent before the end of the trial, usually at the end of each epoch) are fed to an algorithm called assessor.

Assessor

The assessor is an algorithm that evaluates when trials can be stopped early as they are likely not improving the results already obtained from previously tested architectures. This mechanism can be implemented in several ways and is similar to an early stopping mechanism for neural network training. The most common one in NNI is called “MedianStop assessor”, it starts after a certain fixed number of epochs and stops trials that have the performance measure worse than the median performance measure registered in all trials at that point in time. Other assessors are more complex and make use of heuristics to try and predict the outcome of the training of a specific network.

2.4.3 Tuner: Tree-structured Parzen Estimator

The Tree-Structured Parzen Estimator, first introduced in Bergstra et al. (2011), is part of the Bayesian hyperparameter optimization family. The overarching objective of these algorithms is to build a probability model of the objective function by keeping a record of past results evaluated on the validation set. This model is used to select the most promising hyperparameters to test, and the objective function resulting from that test is used to update the probability model.

Sequential Model-Based Optimization (SMBO) algorithms are a natural evolution of Bayesian optimization models and, in a similar fashion, are usually implemented in several applications where evaluating the objective function is expensive in terms of time and computing power. These algorithms use a surrogate model like Bayesian optimization methods do, with the added "sequential" part referring to the trials being tested one after another.

An important component in SMBO methods is the selection function, which is the criteria by which the hyperparameters set drawn in the surrogate model are chosen to be tested on the real model. The most common one, also used in the case of the TPE, is the Expected Improvement, defined as

$$EI_{y^*}(x) = \int_{-\infty}^{y^*} (y^* - y)\, p(y|x) dy, \tag{2.12}$$

where y^* is a threshold value of the objective function, x is the set of hyperparameters, and y is the associated value of the objective function. $p(y|x)$ is the surrogate model used in the hyperparameter optimization experiments.

Different SMBO methods differ in how they construct $p(y|x)$. Specifically, the TPE estimates $p(x|y)$ and $p(y)$ instead of $p(y|x)$ and replaces the distributions of the original configuration with non-parametric densities by using a kernel density estimator. In the TPE, $p(x|y)$ is then expressed in the following terms

$$p(x|y) = \begin{cases} l(x) & \text{if } y < y^* \\ g(x) & \text{if } y \geq y^*, \end{cases} \tag{2.13}$$

where $l(x)$ is the probability density function formed by using the observations $\{x^{(i)}\}$ such that the corresponding loss $f(x^{(i)})$ is less than y^* (a pre-defined threshold) and $g(x)$ is the probability density function formed by all the remaining observations. y^* is usually set at a larger value than the optimal point observed at a given point in time and is a quantile of the observed y values. The optimization thus focuses on drawing hyperparameters which are more likely part of the $l(x)$ function rather than $g(x)$. By applying equation 2.12, the expected improvement results in being proportional to the ratio $\frac{l(x)}{g(x)}$, the TPE algorithm aims at maximizing this ratio in order to optimize the objective function, thus trying to draw values which are part of $l(x)$ with a higher probability compared to $g(x)$.

In practice, the algorithm first needs a defined domain of hyperparameters to create a search space and an objective function that depends on the chosen set of hyperparameters and has to be optimized (minimized/maximized depending on the objective function). It then replaces the distribution of the configuration with non-parametric densities in order to have two density functions $l(x)$ and $g(x)$, as defined in Equation 2.13. Initial samples are randomly selected in order to initialize the algorithm and generate a surrogate model. New hyperparameter sets are then drawn based on the surrogate model from the density function $l(x)$. The objective function of the real model is calculated using the hyperparameter set that, at the time, maximizes the expected improvement measure, the real model's objective function value is then used to update the surrogate model, and the sequence is repeated until a given time budget is exhausted or a limit number of trials is reached.

The crucial difference and reason for good performance when comparing this algorithm with random or grid search is in the history of past results kept in memory and the first evaluation of where to direct the selection of hyperparameters through the optimization of the surrogate model instead of the entire model, which usually needs seconds instead of hours.

Chapter 3

Data and model estimation

3.1 Data Description

Data for the empirical analysis is gathered from the Center for Research in Security Prices (CRSP). Monthly return data, shares outstanding, and prices for US-based stocks are included in the database, as well as further information about their delisting return, share code, and exchange code. The sample period spans from January 1970 to December 2021.

Firm characteristics are gathered from the open-source dataset published in A. Y. Chen and Zimmermann (2021)[2]. I have used version 1.2.0 for this study, which was released in March 2022. The dataset contains 321 firm characteristics reproduced from cross-sectional studies of stock returns, Among these factors, 132 are categorized as clear predictors in both the original publication and the replication. Thus, I use these 132 firm characteristics to train the artificial neural networks and forecast the cross-section of stock returns. A comprehensive list of the firm characteristics used for this study can be found in Appendix B.

The one-month treasury bill rate is used as a proxy for the risk-free rate in order to calculate excess returns. These rates are obtained from the Kenneth French Data Library.[3]

Figure 3.1 shows the correlation matrix between the initial features. For the purpose of training the neural networks part of the empirical analysis, no selection

[2]Data available at www.openassetpricing.com/data

[3]www.mba.tuck.dartmouth.edu/pages/ken.french/data_library

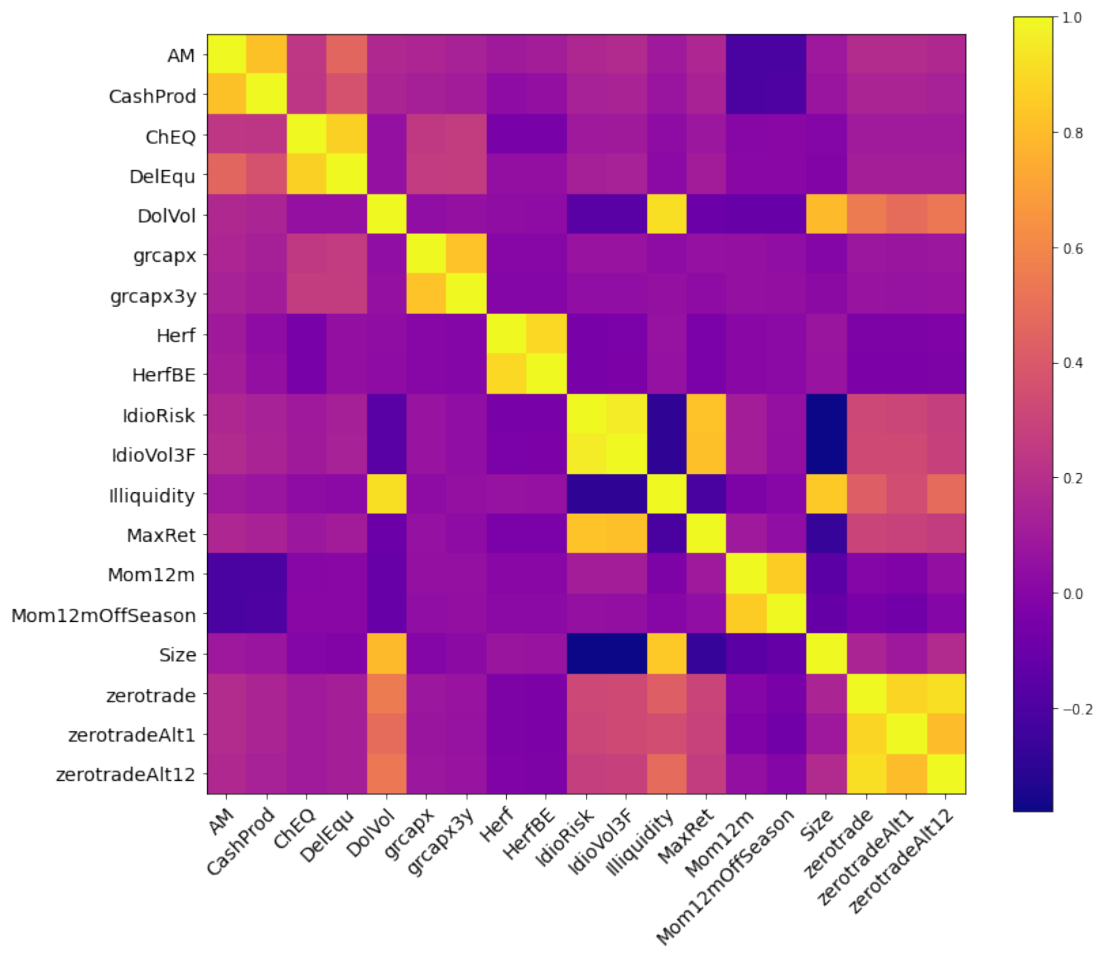


Figure 3.1: Correlation matrix of predictors gathered from the open-source dataset published by A. Y. Chen and Zimmermann, 2021. Only rows that have at least one feature with 80% absolute correlation or higher are included

between highly correlated features has been made since it is thought that these models are not affected by multicollinearity problems given their nature of black-box, over-parametrized models (De Veaux & Ungar, 1994). It is nonetheless visible how the most correlated features are the ones that have multiple definitions, such as Momentum, defined in several slightly different methods and timeframes (see Appendix B).

The initial dataset with stock returns consists of 624 months, from January 1970 to December 2021. On average, there are 6,039 stocks per month. This number becomes much smaller after merging the returns with the firm characteristics and performing the pre-processing described in the following section.

3.1.1 Feature pre-processing

The stocks considered in the analyses are only those with share codes equal to 10 or 11, traded on either the NYSE, AMEX, or NASDAQ exchanges (thus having a share code equal to 1,2 or 3).

All microcap stocks are removed following Fama and French (2007). Microcaps are defined as stocks with a market cap below the NYSE 20th percentile at the end of each June. These are removed for the year until next June, when the size categories are calculated again and updated.

In order to deal with outliers, excess returns are cross-sectionally winsorized at the 90th and 10th percentiles and then de-meaned in each period to preserve cross-sectional information.

Firm characteristics are scaled using rank-based scaling, following Kelly et al. (2019). The scaling is applied as follows:

- For each month, the ranks of the firm characteristics are calculated separately.
- The ranks are divided by the number of non-missing observations of that characteristic in that month
- The value obtained is diminished by 0.5. By doing so all the characteristics take on values in the interval (-0.5, 0.5].
- Stocks that do not have any characteristics for a given month are removed.

- The remaining missing characteristics assume a value imputed using the cross-sectional median of that month's non-missing values (i.e., 0).

The next-month returns are defined as

$$r_{i,t+1} = \frac{p_{i,t+1} - p_{i,t}}{p_{i,t}}, \tag{3.1}$$

where $p_{i,t}$ is the closing price of stock i at the end of month t.

After the pre-processing, the dataset contains an average number of 1,680 stocks per month.

3.2 Training and tuning

3.2.1 Training procedure

The dataset is split into three subsets, a training set, a validation set, and a test set. I follow an expanding window training with rolling validation and prediction in order to perform predictions with past data only and recursively update the model with newly available information.

Specifically, the initial training set spans the first 25 years of the dataset, from 1970 to 1994, the validation set consists of the 5 years subsequent to the end of the training set, and the test set (the data used to predict next-month returns) is always the 12 months following the end of the validation set. After the model has been trained on the current set of data, predictions are made on the test set. The three subsets of data are then updated, and the training set is enlarged to include an additional year of data. In contrast, the validation and test sets are rolled forward in order to comprise, respectively, 5 years and 1 year of data. See Figure 3.2 for a graphic representation of the process.

The forecasting window of 12 months is the result of available computing resources. Even though the model used for predicting the one-month-ahead returns is the same for a given year, features used to make the prediction and weights for the portfolio formation are updated each month (not all features change every month in their real value due to accounting data being updated quarterly or yearly. Nonetheless, the cross-sectionally scaled values may still change).

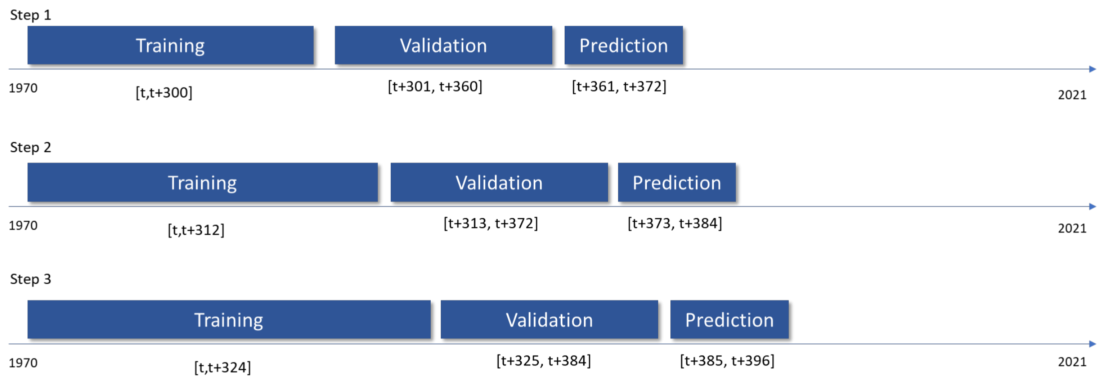


Figure 3.2: The picture shows an example of how the data is recursively split in a training set, a validation set, and a hold-out or test set. t represents January 1970, and the number summed to it represents the number of months after t.

Regularization techniques mentioned in Chapter 2, such as dropout, early stopping, L1 and L2 regularization, are implemented during training (when present in the network hyperparameters set chosen by the tuning algorithm). When performing the returns forecasting with the final networks, after the hyperparameter tuning phase, an ensemble of the predictions is applied with the objective of reducing the variance of the predicted expected returns.

3.2.2 Tuning procedure

The empirical part is split in two parallel experiments: first, I consider a benchmark neural network derived from the published literature and re-train it with the available set of firm characteristics by following the same methodology applied in the original paper. Specifically, I use the best network published in Gu et al. (2020) and perform a grid search around some of the crucial hyperparameters used in their model. By doing so, I derive the initial benchmark model for my research. Secondly, I run an optimization experiment from scratch without using any prior assumptions on the underlying data-generating process, with the objective of automatically finding a good-performing architecture. The search space is broad and contains a comprehensive combination of possible hyperparameter values and network architectures. Both experiments are divided into multiple steps, which are analyzed in more detail in the remaining part of the section.

Grid search using the best performing NN from Gu et al. (2020)

The objective here is to assess whether the results are robust to changing hyperparameters for networks that have already been shown to possess a predictive ability on the cross-section of stock returns. A direct comparison between my results and the benchmark results is not possible as the underlying set of predictors is different. Gu et al. (2020) use a set of macroeconomic variables added to a different set of firm characteristics calculated by the authors. In contrast, I use an external source not containing macroeconomic factors.

Nonetheless, this experiment is helpful in showing whether proven good networks possess robust hyperparameters and, thus, whether small changes in these values lead to disproportionate changes in results. This network also serves as a benchmark to evaluate performance results from a network that has been defined by an algorithm.

I have performed two grid searches in this part. For the first one, smaller, I have tuned four hyperparameters, described in Table 3.1. The possible values are similar to the ones presented in the original publication.

	Low	Mid	High
Learning Rate	0.001	0.005	0.01
L1 λ	1e-5	4.95e-4	1e-3
Adam β_1	0.8	-	0.9
Adam β_2	0.99	-	0.999

Table 3.1: The table shows the lowest and highest possible values part of the search space for each hyperparameter, as well as the mid-point when included. This search space is narrow due to computational resources available, but starts from an already well-defined set of hyperparameters presented in the original publication.

Once the best network from the first search has been identified, a second grid search has been conducted with the learning rate used as the only "tunable" hyperparameter. After this, the best-performing parameters are selected, and an ensemble prediction technique is set up to forecast the cross-section of stock returns and form decile portfolios.

HPO experiment from scratch

This part is where AutoML techniques' potential is assessed, an initial search space is defined, including all possible combinations of hyperparameters affecting the neural network model and its architecture. The experiment is set up using the Tree-structured Parzen Estimator as optimization algorithm. A Median Stop Assessor is implemented to stop all trials which are not at the same performance level as other tested architectures before they are completed. 1000 trials have been conducted in total. Of these, 127 neural networks have been fully trained and tested on the hold-out sample. The others have either failed due to their architecture or most likely stopped early as a better network had already been found and trained. This trial early stopping mechanism is different from the training early stopping technique. The former stops the entire trial, whereas the latter stops training the network for that specified subset of data and moves to the next iteration by shifting the training and validation windows as shown in Section 3.2.1.

Only deep feedforward networks are included in the possible architectures, and alternatives like convolutional neural networks are not included in the research as they are normally not applied to this type of task.

The initial search space consists of the following hyperparameters:

- Number of neurons (can be different for each layer, the maximum number of hidden layers is 5)
- Activation function
- Optimizer
- Learning Rate
- L1 λ
- L2 λ
- Dropout probability
- Batch Normalization (Boolean indicating whether the technique is used or not)

Appendix A contains a complete overview of the possible hyperparameters and the values they can potentially assume during the experiments.

After the first experiment, the best network architecture is selected based on the average Spearman correlation coefficient calculated on the validation set during training. I use this network architecture to perform the following experiments.

First, a grid search along the learning rate axis is performed. 10 equally spaced points (log-based) are selected around the optimal one found in the previous architecture, following Messmer (2017). This is done to assess the robustness of the performance when the learning rate parameter is changed near the optimal point found.

The best learning rate found at this stage is then used for the final prediction part, where an ensemble of the forecasted expected returns is calculated.

The ensemble prediction methodology is implemented in order to reduce the variance from the stochastic component inherent in the process of training neural networks and making forecasts with them. Ideally, this step reduces variance in portfolio returns. 10 different seeds are randomly selected to initialize the training module, and the final return predictions are the average of each forecasted return. Specifically the prediction for a stock i at time t is

$$\hat{r}_{i,t}^{comb} = \frac{1}{10} \sum_{n=1}^{10} \hat{r}_{n,i,t}^{single} \tag{3.2}$$

3.3 Forecast evaluation

There are two main ways in which the results are evaluated, depending on the analysis stage. During the first part, while the network is being searched and selected, the performance is assessed through statistical measures. In the final part, after the ensemble predictions are completed, the performance measures are those related to portfolio returns.

3.3.1 Statistical measures

The statistical performance measures used in this thesis are primarily two. The Spearman rank correlation coefficient, also called rank correlation in the following, and the Mean directional accuracy, with the added cost function used during the training phase of each neural network.

The Mean Directional Accuracy measure is defined as

$$\mathrm{MDA} = \frac{1}{T}\sum_{t=1}^{T} \mathbb{1}_{\mathrm{sgn}\left(r_t - r_{t_1}\right)=\mathrm{sgn}\left(\hat{r}_t - r_{t-1}\right)}, \tag{3.3}$$

where sgn is the sign function and $\mathbb{1}$ is the indicator function, $\hat{r}_t$ is the forecasted return at time t and r_t is the realized return at time t.

The rank correlation coefficient is measured as

$$\mathrm{Spearman} = 1 - \frac{6\sum d_i^2}{n\left(n^2-1\right)}, \tag{3.4}$$

where n is the number of data-points considered, and d_i^2 is the square of the difference in the ranks of $r_{i,t}$ and $\hat{r}_{i,t}$.

Both measures are used as a way to assess the statistical performance of neural networks when calculated on the hold-out sample, additionally, in the intermediate steps of experiments, the Spearman correlation is calculated on the validation set, and its average is used to assess the performance of architectures and select the ones to use for forecasting expected returns.

3.3.2 Portfolio construction and evaluation

After monthly return predictions are made, these are split into 10 deciles. The highest decile (long portfolio) and the lowest decile (short portfolio) are combined to form a long-short portfolio. This portfolio is formed for the benchmark network (BNN) and all the fully-trained networks found during the HPO experiment.

The first measures used for the comparison are the cross-sectional average return, standard deviation, and Sharpe Ratio of the three models.

The maximum drawdowns (maximum loss from a peak to a trough of a portfolio) of each network are calculated as follows

$$MaxDD = \max_{0\leq t_1\leq t_2\leq T}\left(\frac{1+R_{t_1}}{1+R_{t_2}} - 1\right), \tag{3.5}$$

where R_t is the cumulative return from date 0 through t.

The average monthly turnover is calculated for the long and short portfolios, and defined following Gu et al. (2020) as

$$TN = \frac{1}{T}\sum_{t=1}^{T}\left(\sum_{i}\left|w_{i,t+1} - \frac{w_{i,t}\left(1+r_{i,t+1}\right)}{1+\sum_{j} w_{j,t}r_{j,t+1}}\right|\right), \tag{3.6}$$

where $w_{i,t}$ is the weight of stock i in the portfolio at time t.

For the long-short portfolio strategy, turnover is defined as

$$TN^{LS} = TN^{L} + TN^{S}, \tag{3.7}$$

where the L in TN^{L} indicates the long portfolio and the S indicates the short portfolio, both measures are calculated following Equation 3.6.

Moreover, a regression of the portfolio returns on the Fama-French 5 Factor model, augmented with the Momentum and Short-Term reversal factors, is performed. The existence of an α is checked, as well as its t statistic. The loadings on each factor are analyzed to evaluate the characteristics of the portfolios formed by the neural networks.

Following Grinold and Kahn (1999, p.327), I calculate the Information Ratio (IR) using the above-mentioned factor model as benchmark through the relation

$$IR \approx \frac{\text{t-stat}}{\sqrt{T}}, \tag{3.8}$$

where t-stat is relative to the α of the neural network on the factor model. Citing again Grinold and Kahn (1999): "This relationship becomes more exact as the number of observations increases". It is thus possible to substitute the "approximately equal" sign in Equation 3.8 with an "equal" sign given the large number of years considered in the hold-out sample, 21.

Chapter 4

Results

The chapter is divided in three parts. Section 4.1 shows the results of the search experiments, including the best and worst networks found, the best benchmark network, and the behavior of these neural networks when one hyperparameter, in this case the learning rate, is slightly perturbed. Section 4.2 describes the statistical performance of the models over the hold-out sample, including the benchmarks, the best and worst networks, and the average results of all experiments. Section 4.3 provides an insight into the most important features based on which the neural networks have been making their predictions, and Section 4.4 shifts into portfolios formed with the forecasted expected returns, highlighting their performance compared to the benchmark neural network and a factor model.

4.1 Search experiments results

4.1.1 Grid search with benchmark network

The first grid search performed with the benchmark four layers network consists in tuning the 4 main parameters listed in Section 3.2.2. The resulting set of hyperparameters chosen for the second part of the analysis, based on the average best validation performance of each set, is the following:

- L1 λ: 1e-5
- Adam β_1: 0.9

- Adam β_2: 0.999

The subsequent grid search is performed with the learning rate as the only tunable hyperparameter, given that it is considered one of the most sensitive when tuning a neural network. 10 possible values are directly derived from Gu et al. (2020)'s original tuning range, and the neural network is trained each time with a different learning rate. Figure 4.1 shows a relatively stable value of out-of-sample rank correlation, variability in performance is expected, especially with the large difference between each trial in terms of learning rate value. To a certain extent, the changes in the learning rate in this search are of a much larger magnitude than often considered in neural network hyperparameter tuning processes. This choice is done in order to minimize the total number of trials performed, due to computational constraints.

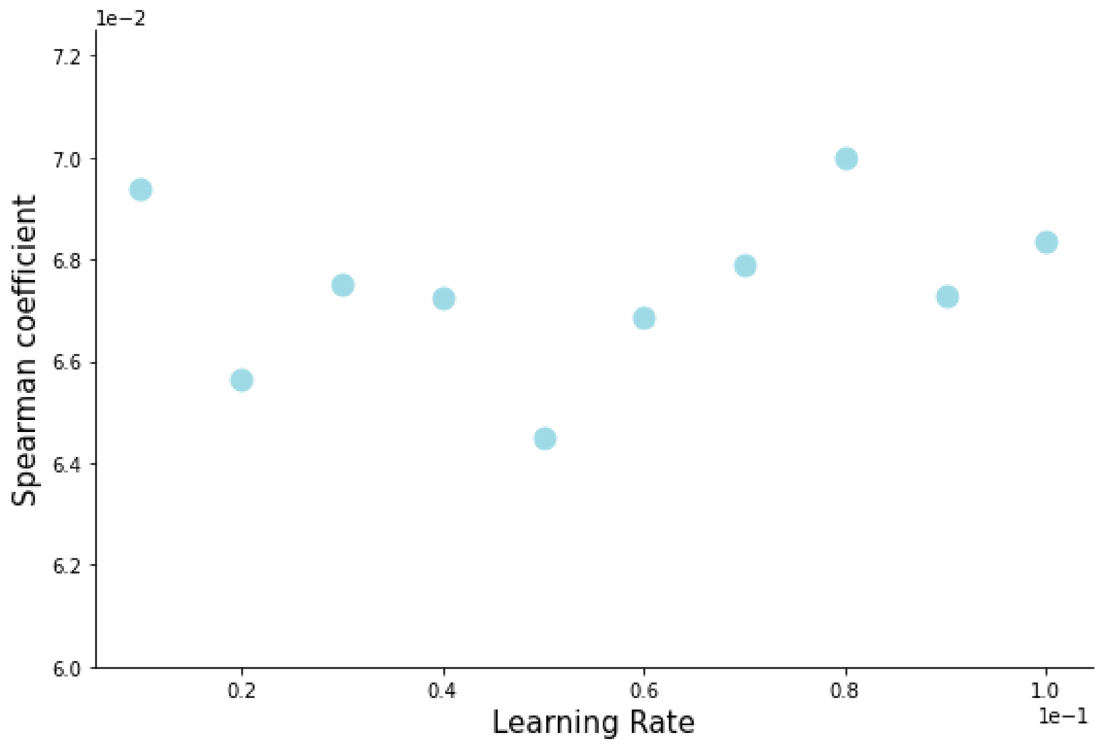


Figure 4.1: The scatter plot shows how the out-of-sample rank correlation varies by varying the learning rate parameter in the identified benchmark neural network.

When performing the final ensemble prediction, the network with the best average validation Spearman coefficient is chosen, in this case, the one with a learning rate of 0.1, which is interestingly not the one with the highest out-of-sample rank correlation.

4.1.2 HPO experiment from scratch

The complete search experiment counts 1000 trials; 127 networks have been fully trained and tested on the hold-out set. The others have been automatically stopped from the Median-based early stopping assessor described in Section 2.4.2, for having, after a pre-defined number of epochs (set at 100 for this experiment), intermediate loss function values worse than the overall median loss at that moment. Table 4.1 shows the main hyperparameters and architectures of the best 5 and worst 5 fully trained networks. A detailed description of these architectures can be found in Appendix C. It is possible to see how there is a relatively marked dispersion in results between the top- and worst-performing networks, even though this is not reflective of the real underlying distribution generated from the optimization routine, as per the reasons stated above regarding the training early stoppage of worse-than-median networks.

The top-performing network is selected for the second part of the experiment, a grid search of 10 equally spaced (log-based) learning rate values around the previously found optimal one to fine-tune the parameter and roughly assess its robustness. The analysis results are shown in Figure 4.2. It is possible to see a pattern in the network's performance, with close performances for close learning rate values. This highlights an interesting result, which would need to be analyzed more in-depth to have conclusive opinions, but suggests that a good-performing network found through an optimization routine can be a robust network that resists small perturbations of one of its most important parameters.

Similarly to the benchmark network, the best-performing model on the validation set based on rank correlation is selected for the ensemble forecasting.

4.2 Model

4.2.1 Predictions Statistics

After the tuning phase, the best-performing networks are tested on the hold-out set, and statistical performance measures are calculated on the forecasted expected returns. Table 4.2 shows the statistical performance of the networks found by the AutoML algorithm and the benchmark models, including a simple

	CORR	HL	LR	L1	L2	Drop	BN	OPT	ACT
Top 5 networks									
T1	7.16	3	0.1	1	0	0.7	1	Nadam	ReLU
T2	6.87	5	0.88	0.001	0.578	0.16	1	Nadam	LeakyReLU
T3	6.79	4	0.303	0.036	1.42	0.08	0	Nadam	LeakyReLU
T4	6.76	3	0.1	0.001	1	0.701	0	Adam	ReLU
T5	6.72	5	0.08	0.22	1.91	0.11	0	Adam	LeakyReLU
Bottom 5 networks									
B5	-0.72	5	0.026	0.063	0.016	0.15	0	Adadelta	LeakyReLU
B4	-1.14	5	0.166	0.012	0.384	0.42	0	Adadelta	LeakyReLU
B3	-1.31	5	0.079	0.519	0.047	0.269	0	Adagrad	ReLU
B2	-1.42	5	0.003	0.078	7.73	0.398	0	Adadelta	LeakyReLU
B1	-1.56	4	0.077	0.056	0.031	0.27	0	Adadelta	LeakyReLU

Table 4.1: This table shows a selection of the hyperparameters of the top 5 and bottom 5 performing networks found during the optimization experiment. CORR is the out-of-sample monthly Spearman Correlation coefficient for the network in percentage points, LR is the learning rate, L1 and L2 are the λs used respectively for L1 and L2 regularization, these three are all multiplied by a factor of 1000. HL is the number of hidden layers, BN shows whether batch normalization has been used or not in the network (1 = True, 0 = False), OPT and ACT are the optimization algorithm and the activation function of the networks

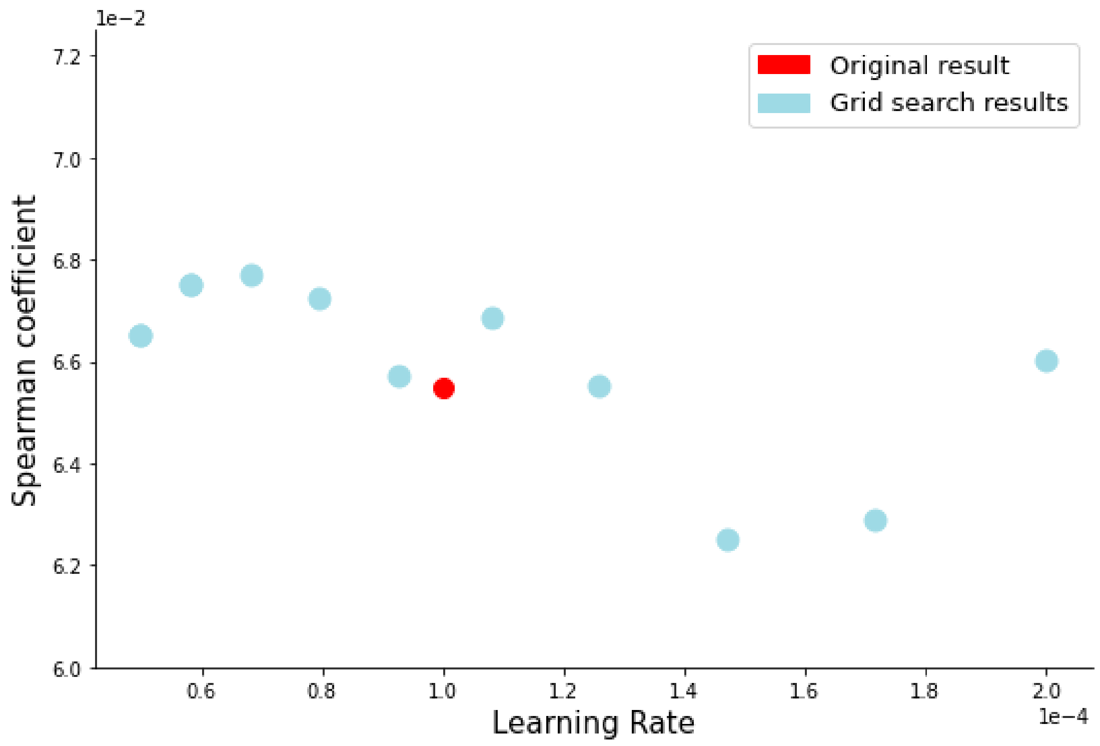


Figure 4.2: The scatter plot shows how the out-of-sample rank correlation varies by varying the learning rate parameter in the identified benchmark neural network.

Pooled OLS regression used as the linear benchmark. The worst found network, as well as the mean values and standard deviation of all the experiment results, show dispersion in the results, demonstrating that the optimization routine has identified hyperparameter combinations that improved on the previous trials with the advancement of the experiment. Interestingly, the top-performing network performs similarly to the benchmark network but achieves a lower rank correlation. Both neural networks perform better than the linear benchmark. The average of the experiment, on the other hand, stands at a much lower value than the linear benchmark, showing that even the best-than-median fully trained networks were, on average, performing at a mediocre level and that it is not easy to tune a neural network even just to match the performance of a linear model on a cross-sectional study such as this one.

	CORR	Direction(%)
Top Network (TNN)	7.2	52.4
Worst Network (WNN)	-1.56	49.5
Benchmark NN4 (BNN)	7.66	52.48
Linear Benchmark	5.67	51.9
Experiment		
Average	3.71	51.3
Std. dev.	2.76	0.96

Table 4.2: The table shows the out-of-sample Spearman rank correlation coefficient (CORR) and the Mean Directional Accuracy (MDA), both in percentage points, for the benchmark network, a linear benchmark, the best and worst networks found in the experiment, as well as the average and standard deviation of all the networks trained during architecture search.

4.2.2 Returns Distribution

Figure 4.3 shows a comparison of the returns distribution generated by the predictions of three neural network models, TNN, WNN, and BNN, with the actual distribution of the stock returns. As expected, real returns show a wide distribution, with "fat tails" around the ±10% mark. The returns predicted from the three models are closer to 0, especially WNN, which underperforms all tested models. Regarding the other two networks, BNN has a distribution skewed to the left, while TNN is slightly more skewed to the right. Still, both networks predict returns near the value of 0. The reason could be attributed to the set of features having a small range of values they take before being fed to the neural networks.

4.3 Feature importance

There are several methods to assess the importance of input features in a neural network, such as Local Interpretable Model-agnostic Explanations (LIME), Shapley Additive Explanations (SHAP), and Individual Conditional Expectation (ICE). These are called model-agnostic methods, given that they can be applied to several types of machine learning models. Some recently developed methods introduced in the literature are specific to Deep Neural Networks, such as gradient-

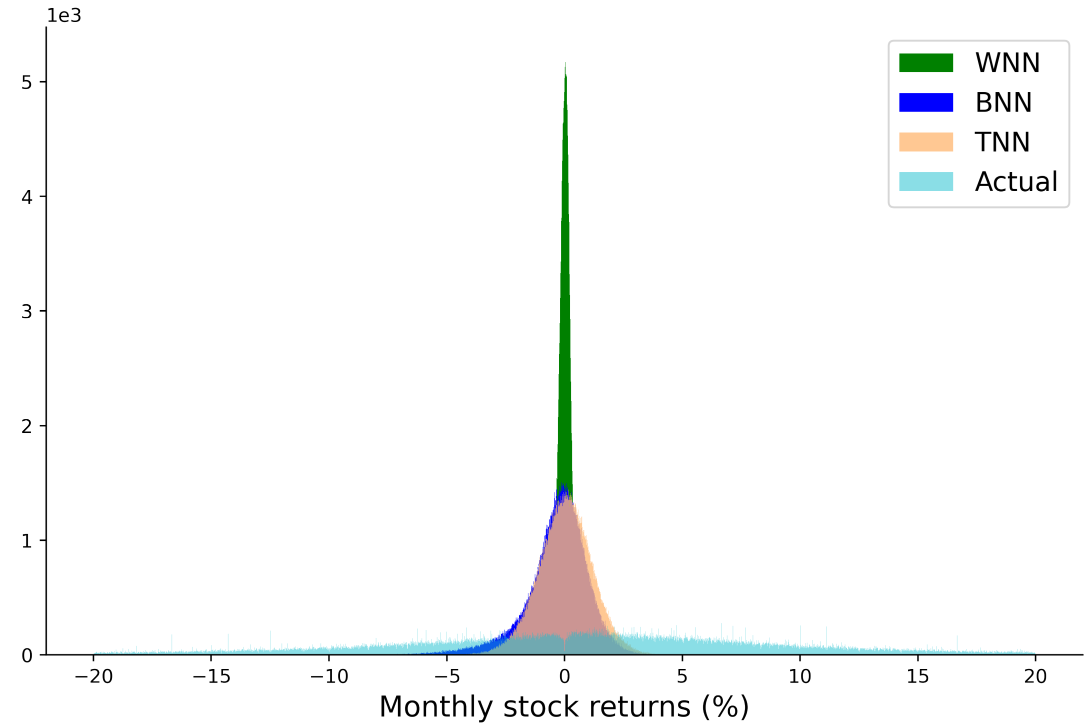


Figure 4.3: The image shows the distribution of the predicted returns for three models, the top network found in the experiment (TNN), the worst one (WNN), and the benchmark network (BNN). These three are compared to the distribution of realized returns.

based methods. These take advantage of the neural networks' structure, which is the same that enables gradient-based optimization (backpropagation). The most common algorithm in this last category is called Integrated Gradients, first published in Sundararajan et al. (2017). This is the method used in this analysis for assigning an importance score to each firm characteristic with regard to the prediction of stock returns. The analysis is conducted for the two best networks found during the empirical research, TNN and BNN.

On a high level, a greater value of the mean Integrated Gradient score for a given feature indicates greater importance for that predictor. The magnitude of each score is related to the strength of the contribution provided to the predicted value. Intuitively, if a feature with negative score is deleted, the prediction should be higher, and the opposite holds true for a feature with a positive score.

I have used the Integrated Gradients method implemented in the Python library Captum[4]. Figure 4.4 shows the average Integrated Gradients scores of the firm characteristics for TNN. Only the highest 20 values in absolute terms are shown in the chart, while a complete picture is provided in Appendix D. The most important features include Assets-to-Market (AM), Dividend seasonality (DivSeason), Trend Factor (TrendFactor), and Return on assets, quarterly updated (roaq). It is interesting to note how these represent, respectively, a value factor, a payout indicator, a momentum factor, and a profitability factor. All of these are, on average, increasing the predicted return when they increase, which makes intuitive sense. The same holds true for the larger negative scores, OScore, which is a proxy for bankruptcy risk, IdioVolAHT and IdioVol3F, both of which represent the standard deviation of the residuals from regressions on the CAPM and Fama-French 3-factor model, respectively.

The most important features of BNN are slightly different compared to the previous one. Figure 4.5 shows the Integrated Gradients score for the top 20 features in absolute value. The first evident difference compared to TNN is the presence of a significant number of large negative scores in the top 20. The largest positive scores include Book to market with December market equity (BMdec), Assets to market (AM), Short Term Reversal (STreversal), 12-month momentum (Mom12m), and the Trend Factor (TrendFactor). Two value factors are thus present in the top 5, BMdec and AM, and three momentum factors. Other cat-

[4]captum.ai/api/integrated_gradients

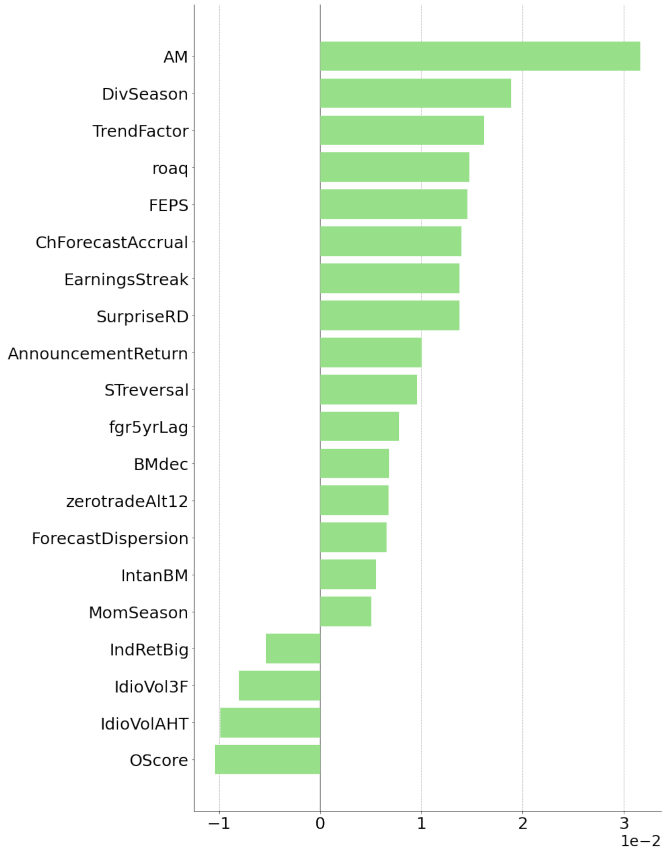


Figure 4.4: The chart shows the average feature importance calculated for the best-performing network (TNN) using the Integrated Gradients method.

egories like quality, investment, and profitability are still included, even though with a smaller contribution. It is not intuitive how the STreversal factor may be positively correlated with higher predicted returns. This may be due to two reasons at first glance. First, the signs of the factor are not necessarily the same as the original definition after the data pre-processing (firm characteristics are cross-sectionally scaled between -0.5 and 0.5), or the network may be misusing the feature.

Amongst the largest negative scores, it is possible to see Idiosyncratic volatility based on CAPM (IdioVolAHT), similar to TNN, days with zero trades in the past month (zerotradeAlt1), the maximum daily return over the past month (MaxRet), and the 12-month change in inventory divided by average total assets (ChInv).

Figure 4.6 contains some exemplary scores of the two networks analyzed in this section, split by firm characteristic category. Regarding profitability factors, it

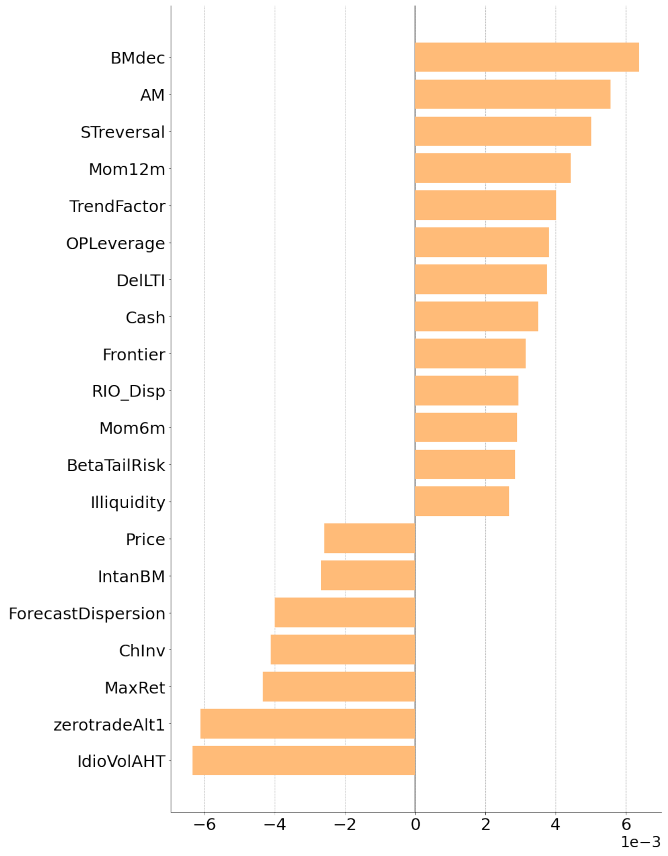
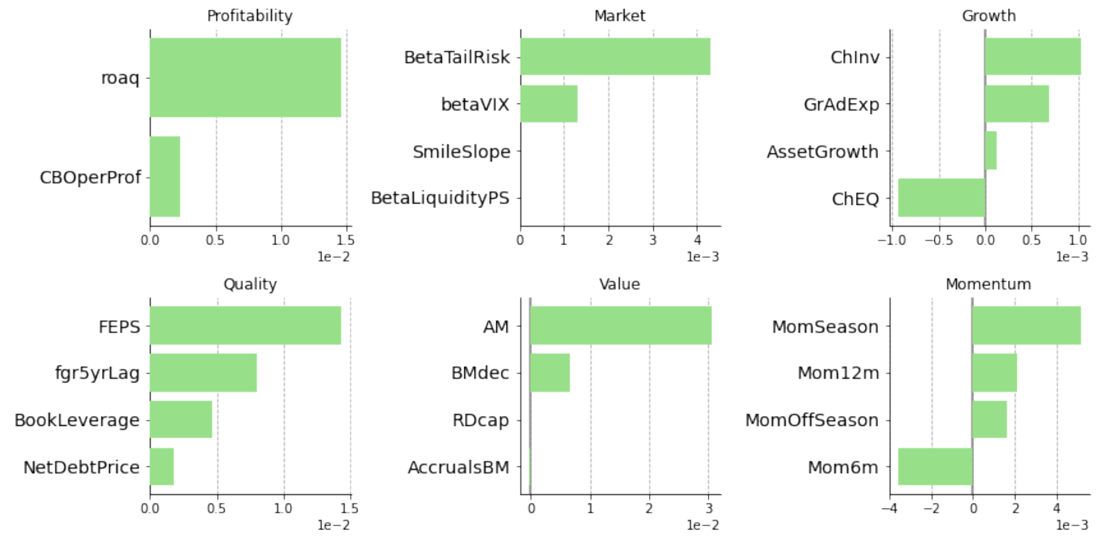


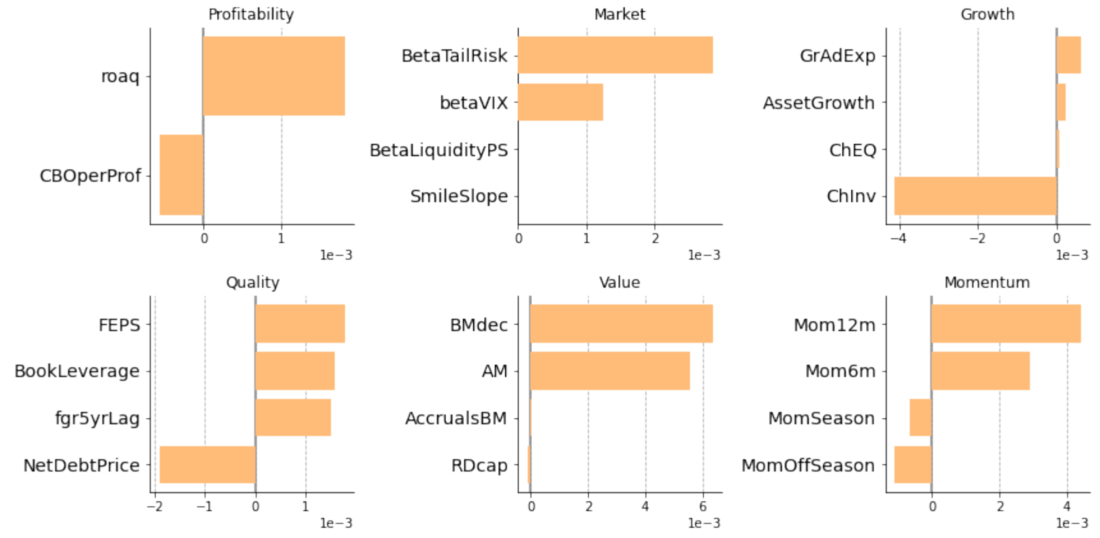
Figure 4.5: The chart shows the average feature importance calculated for the benchmark neural network (BNN) using the Integrated Gradients method.

is interesting to note that operating profitability with working capital and R&D adjustments (CBOperProf) has two opposite signs in the two networks, positive for TNN and negative for BNN. The same goes for ChInv in growth factors, net debt to price (NetDebtPrice) in quality factors, and all the included momentum factors except 12-month momentum. Value factors are somewhat similar in contribution to the network in terms of sign and ranking, even though with different score magnitudes. Most of the time, factors with small or zero scores for TNN have the same or similar value for BNN, too. Examples are R&D capital-to-assets (RDcap), AccrualsBM, the Pastor-Stambaugh liquidity beta (BetaLiquidityPS), the put option volatility of the stock minus its call option volatility (SmileSlope), and others not shown in Figure 4.6.

Deeper analyses on the importance of each firm characteristic in the predictions made by deep neural networks and how these weights change across time is a point that may be interesting for future research, holding the potential to provide more insights on the over-performance of this kind of algorithms over other machine learning methods and especially over linear models.



(a) Feature importance of TNN model



(b) Feature Importance of BNN model

Figure 4.6: The plots show the average feature importance for the TNN (green) and BNN (orange) models, calculated using Integrated Gradients for several firm characteristics, split by category.

4.4 Portfolio performance

4.4.1 Returns Analysis

Portfolios are formed for the benchmark network and all the fully-trained networks found during the HPO experiment. For these last ones, I take the average of each performance measure and its standard deviation and separately show the best-performing network and the worst-performing one. Regarding portfolio returns, I show how there is a discrepancy between the top- and worst-performing networks found through the experiment, similarly to statistical prediction performance. The optimization routine has thus been able to improve its predictions over time and find a network performing better than the average on the test set. Even in this case, the average portfolio performance of all the tested networks is skewed towards the best-performing network results given the trial early stopping algorithm. Due to the design decisions taken for this empirical analysis, it is impossible to have a clear view of the worst-performing networks as it has been deemed unnecessary to spend computational resources and time training them.

Table 4.3 contains the most important measures on long-short value-weighted portfolios formed with the previously listed models. Similarly to statistical performance measures, BNN performs slightly better than TNN. BNN presents a lower maximum drawdown of 40% compared to 47.5% of TNN, which is closer to the average of all networks. Turnover is also very similar between the two, while it is slightly lower for the average network performance and even more moderate in WNN. The important notable differences are in mean returns and Sharpe Ratios. BNN tops the list with a 1.85% average monthly return, around 1% higher than the average experiment. WNN has a negative average return over its testing period, causing the maximum drawdown to be almost 100%. Sharpe Ratio is very similar between BNN and TNN, slightly higher in TNN due to a lower volatility of returns. Both BNN and TNN show a significant α over the benchmark factor model considered. WNN has no α while the average portfolio shows a significant α of 0.63%. Information Ratios are again very similar between BNN and TNN and much smaller in the other two portfolios. The R^2 resulting from the regression is not high in any of the portfolios. The portfolio that the “7-factor” model best explains is BNN, with 40.9% of the variation explained. TNN stands at an even lower value of 22.8%, indicative of the purely optimization-wise mechanisms that are part of the design of TNN. At the same time, other considerations have surely been made by the authors of the original publication for which BNN was designed.

	BNN	TNN	WNN	Experiment	
				Avg	Std
MaxDD(%)	40.01	47.5	100	47.2	74.02
Max1MLoss(%)	13.45	9.05	10.81	10.53	5.83
Turnover long (%)	151.32	143.24	77.69	108.33	33.44
Turnover short (%)	124.29	131.1	71.81	102.82	29.36
Turnover(%)	275.6	274.34	149.5	211.15	61.86
Mean ret.(%)	1.85	1.51	-0.1	0.74	0.65
Std. Dev.	4.96	3.98	3.43	3.12	1.51
SR	0.37	0.38	-0.03	0.21	0.16
FF5+Mom+STrev α	1.58	1.39	0.01	0.63	0.59
t(α)	6.47	6.18	0.07	3.12	2.55
R^2	40.9	22.77	31.8	27.74	17.8
IR	0.36	0.34	0.004	0.18	0.14

Table 4.3: This table shows the main performance statistics of the long-short portfolios formed with the optimal network found through the optimization experiment (TNN) and a benchmark network from Gu et al., 2020 (BNN). The worst network found is also included (WNN) and, for reference, the average of portfolios formed through all the networks fully trained during the experiment is present, as well the standard deviation of the values. The portfolio statistics are calculated on monthly portfolio returns and include the Maximum Draw-down, the maximum 1-month loss as well as turnover measures, average returns, their standard deviations, and Sharpe Ratio. Measures deriving from regressing portfolio returns on a factor model, defined as the Fama-French 5 Factor model augmented by Momentum and Short-Term Reversal, are present, namely the α over the factor model, the corresponding t-stat, Information Ratio (IR), and the R^2 of the regression.

Figure 4.7 shows the cumulative log returns of long-short portfolios compared to the value-weighted cumulative market log returns. It is interesting to note how the returns are slightly dominated by the short side, given the networks' better ability at forecasting lower-decile stocks compared to top-decile stocks at each

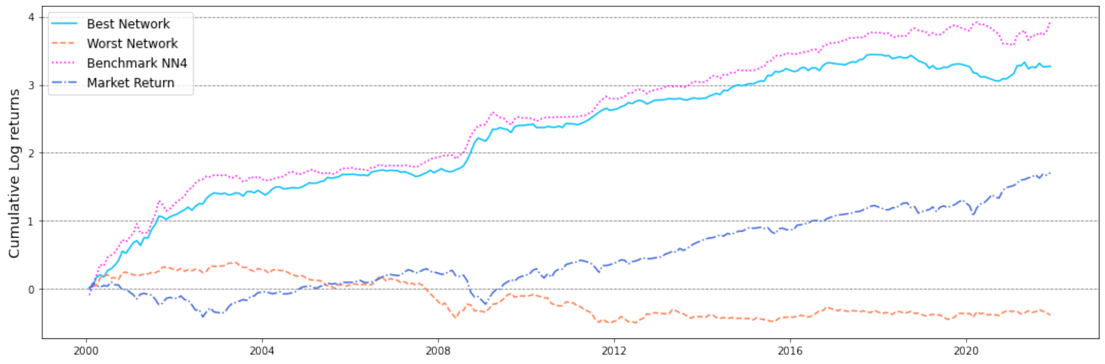


Figure 4.7: Long-short portfolio cumulated returns shown in logarithmic form from January 2000 to December 2021.

given month.

4.4.2 Factor-based Analysis

When regressing the long-short portfolio returns of the two analyzed networks on the Fama French 5 Factor Model augmented with Momentum and Short-term Reversal, the results are interesting and worth a deeper analysis. Table 4.4 and Table 4.5 show the detailed results of the long-short portfolio regressions for TNN and BNN, respectively. The results are similar for the two networks; both present a statistically significant α, already shown in the previous section, representing an over-performance of the portfolio over the benchmark model. Both portfolios have a slightly negative loading on the SMB factor, meaning that the portfolios behave like large-cap portfolios. On the other hand, a significantly positive HML loading, around 0.3 or higher for both models, denotes that these portfolios behave like value portfolios. The other loadings are not significantly different from 0 in the case of TNN. Thus, nothing can be explicitly concluded about these factors. On the other hand, BNN shows a significantly positive loading on the quality factor (RMW), meaning that the portfolio co-varies with high-quality (robust) portfolios, and a significantly positive loading on the investment factor (CMA), pointing out a low investment-like portfolio. The loading on the Momentum (Mom) factor is significantly positive but still small, standing at 0.13.

As expected, the loading on Short-Term reversal is negative for both networks but not significantly different from 0.

	Coef.	std. err.	t	$P > \|t\|$	[0.025	0.975]
const	1.39	0.23	6.18	0	0.95	1.83
Mkt-RF	-0.04	0.06	-0.77	0.44	-0.16	0.07
SMB	-0.16	0.08	-2.07	0.04	-0.31	-0.01
HML	0.35	0.1	3.67	0	0.16	0.53
RMW	0.14	0.1	1.4	0.16	-0.06	0.33
CMA	0.26	0.14	1.89	0.06	-0.01	0.53
Mom	0.02	0.05	0.38	0.71	-0.08	0.11
ST_Rev	-0.01	0.062	-0.11	0.92	-0.13	0.12

Table 4.4: The table shows the results of the value weighted long-short portfolio excess returns, forecasted by the best neural network model found in the experiments (TNN), regressed on the Fama-French 5 Factor model augmented with Short-term reversal and Momentum. The coefficient, the standard error (std.err.), t-statistic (t), p-value (P $>$ $|t|$) and confidence intervals are shown for each parameter

	Coef.	std. err.	t	$P > \|t\|$	[0.025	0.975]
const	1.58	0.25	6.47	0	1.1	2.068
Mkt-RF	-0.04	0.06	-0.66	0.5	-0.17	0.084
SMB	-0.46	0.08	-5.52	0	-0.62	-0.29
HML	0.28	0.1	2.77	0.006	0.082	0.49
RMW	0.43	0.11	4.01	0.001	0.22	0.65
CMA	0.34	0.15	2.25	0.03	0.04	0.63
Mom	0.13	0.05	2.53	0.01	0.03	0.24
ST_Rev	-0.13	0.07	-1.87	0.06	-0.26	0.007

Table 4.5: The table shows the results of the value weighted long-short portfolio excess returns, forecasted by the benchmark neural network model (BNN), regressed on the Fama-French 5 Factor model augmented with Short-term reversal and Momentum. The coefficient, the standard error (std.err.), t-statistic (t), p-value (P $>$ $|t|$) and confidence intervals are shown for each parameter

Chapter 5

Conclusion and Future Research

5.1 Summary of Results

In this thesis, I explore one of the emerging strands of Machine Learning, AutoML, and try to understand whether the techniques being developed for this scope apply to the tuning and training of Deep Neural Networks in the context of forecasting the cross-section of stock returns. Automatic optimization of neural network design and hyperparameters has usually been conducted with algorithms like Grid Search and Random Search, or a combination of the two. This process is generally inefficient in terms of time and computing power allocated to complete it. Using one of the optimization algorithms recently developed for this purpose, the Tree Parzen-Estimator, I try to demonstrate how given moderate amounts of time and computing power budgets, good-performing neural networks can still be automatically found and trained.

The actual results of this implementation can drastically vary depending on the resources available for the experiments. This thesis aims to check whether there is a potential for AutoML techniques to be implemented in the financial setting, and this has been shown to be existing.

This research has confirmed that it is possible to forecast the cross-section of stock returns through deep neural networks designed and tuned automatically from an optimization algorithm without the assumptions injected by a subject matter expert. As expected, among all the trained networks, there is substantial variability in results. Even small changes in a single hyperparameter can generally lead to important changes in prediction performance. Thus, it is expected to see some bad-performing networks during the optimization process. In this case, the majority of selected networks, especially during the first phases of the experiment, were either stopped due to their intermediate performance being worse than the median performance of all trials at that point or were unsuccessful in performing predictions. A notable result is that the networks found did not perform at the level of the pre-selected benchmark network, already published in the literature, even though this model has been adapted and re-trained on a different dataset, containing more predictors and without macroeconomic factors being included. This goes in favor of the hypothesis that considers a combination of both automatic and manual architecture design and hyperparameters tuning to be optimal in order to find good-performing and robust deep neural networks that are able to forecast the cross-section of stock returns and can eventually be re-trained and adapted to additional or different predictors.

Another critical insight that can be drawn from the experiment is the fact that the optimization routine has been able to find a neural network that can be defined as robust. A small robustness check on the learning rate hyperparameter has been used as a test. It was conducted on the network that showed the best correlation coefficient on the validation set (TNN) and measured through the out-of-sample correlation coefficient. As expected, similar results were found for the benchmark network (BNN).

The two models used for the final stages of the analyses, BNN and TNN, showed a similar set of features amongst the most important ones contributing to the predicted returns, according to the Integrated Gradients scores calculated, even though some features representing crucial categories according to cross-sectional forecasts literature have been exploited in opposite directions from the two networks, indicating misbehavior of either one of them depending on the single case.

5.2 Future research

This kind of research could be continued and expanded in several directions. First of all, the over-performance of deep neural networks in cross-sectional studies has now been shown multiple times in the literature and can be given as granted under the right set of assumptions; the next step would now be to implement methodologies focused on automating architecture search and hyperparameter optimization with more sophisticated neural networks in order to use their full potential. Creating more complex models could lead to even better performance levels than the current one; one example could be the exploitation of alternative sources of data, such as textual data (i.e., news reports, Twitter feeds, etc..) or images, such as satellite imagery data for manufacturing or logistic companies in order to expand Deep Feedforward Neural Networks with Convolutional Neural Networks and Recurrent Neural Networks.

Other future ideas could include the test of a similar set of assumptions as the ones used in this thesis and change the task from a regression problem to a classification problem, specifically by splitting returns in deciles before forecasting the expected returns and by using the decile itself as the category to be forecasted through the prediction at a cross-sectional level. This has been tried in some publications such as Abe and Nakagawa (2020) but not with the implementation of automated techniques such as the ones used in this study.

This empirical research has been conducted with a powerful machine composed of only CPUs. It is widely known how GPUs drastically improve neural networks' performance in terms of training times. The results of the kind of experiment conducted in this thesis can be improved if the time budget or the computing power available are larger. Nonetheless, as mentioned, the principle and the feasibility of such processes remain demonstrated by this study.

Bibliography

Abe, M., & Nakagawa, K. (2020). How Do We Predict Stock Returns in the Cross-Section with Machine Learning? *2020 3rd Artificial Intelligence and Cloud Computing Conference*, 63–69. https://doi.org/10.1145/3442536.3442547

Bergstra, J., Bardenet, R., Bengio, Y., & Kégl, B. (2011). Algorithms for Hyper-Parameter Optimization. *Advances in Neural Information Processing Systems*, *24*. Retrieved November 13, 2022, from https://proceedings.neurips.cc/paper/2011/hash/86e8f7ab32cfd12577bc2619bc635690-Abstract.html

Bergstra, J., & Bengio, Y. (2012). Random search for hyper-parameter optimization. *The Journal of Machine Learning Research*, *13*, 281–305.

Bischl, B., Binder, M., Lang, M., Pielok, T., Richter, J., Coors, S., Thomas, J., Ullmann, T., Becker, M., Boulesteix, A.-L., Deng, D., & Lindauer, M. (2021, November 24). Hyperparameter Optimization: Foundations, Algorithms, Best Practices and Open Challenges. https://doi.org/10.48550/arXiv.2107.05847

Chen, A. Y., & Zimmermann, T. (2019, August 19). Publication Bias and the Cross-Section of Stock Returns. https://doi.org/10.2139/ssrn.2802357

Chen, A. Y., & Zimmermann, T. (2021, May 21). *Open Source Cross-Sectional Asset Pricing* (SSRN Scholarly Paper No. 3604626). Social Science Research Network. Rochester, NY. https://doi.org/10.2139/ssrn.3604626

Chen, L., Pelger, M., & Zhu, J. (2019). Deep Learning in Asset Pricing. *SSRN Electronic Journal*. https://doi.org/10.2139/ssrn.3350138

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, *2*(4), 303–314. https://doi.org/10.1007/BF02551274

De Veaux, R. D., & Ungar, L. H. (1994). Multicollinearity: A tale of two nonparametric regressions. In P. Cheeseman & R. W. Oldford (Eds.), *Selecting*

Models from Data (pp. 393–402). Springer. https://doi.org/10.1007/978-1-4612-2660-4_40

Domashova, J. V., Emtseva, S. S., Fail, V. S., & Gridin, A. S. (2021). Selecting an optimal architecture of neural network using genetic algorithm. *Procedia Computer Science*, *190*, 263–273. https://doi.org/10.1016/j.procs.2021.06.036

Fama, E. F., & French, K. R. (1992). The Cross-Section of Expected Stock Returns. *The Journal of Finance*, *47*(2), 427–465. https://doi.org/10.2307/2329112

Fama, E. F., & French, K. R. (1993). Common risk factors in the returns on stocks and bonds. *Journal of Financial Economics*, *33*(1), 3–56. https://doi.org/10.1016/0304-405X(93)90023-5

Fama, E. F., & French, K. R. (2007, June 1). Dissecting Anomalies. https://doi.org/10.2139/ssrn.911960

Fama, E. F., & French, K. R. (2014, September 1). *A Five-Factor Asset Pricing Model* (SSRN Scholarly Paper No. 2287202). Social Science Research Network. Rochester, NY. https://doi.org/10.2139/ssrn.2287202

Fama, E. F., & French, K. R. (2015). A five-factor asset pricing model. *Journal of Financial Economics*, *116*(1), 1–22. https://doi.org/10.1016/j.jfineco.2014.10.010

Feng, G., Polson, N., & Xu, J. (2018). Deep Learning Factor Alpha. *SSRN Electronic Journal*. https://doi.org/10.2139/ssrn.3243683

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

Goodfellow, I., Vinyals, O., & Saxe, A. M. (2015, May 21). Qualitatively characterizing neural network optimization problems. https://doi.org/10.48550/arXiv.1412.6544

Gridin, I. (2022). Introduction to Neural Network Intelligence. In *Automated Deep Learning Using Neural Network Intelligence* (pp. 1–30). Apress. https://doi.org/10.1007/978-1-4842-8149-9_1

Grinold, R., & Kahn, R. (1999). *Active portfolio management: A quantitative approach for producing superior returns and selecting superior returns and controlling risk*. Mcgraw-hill. https://books.google.it/books?id=a1yB8LTQnOEC

Gu, S., Kelly, B., & Xiu, D. (2020). Empirical Asset Pricing via Machine Learning. *The Review of Financial Studies*, *33*(5), 2223–2273. https://doi.org/10.1093/rfs/hhaa009

Hanauer, M. X., & Kalsbach, T. (2022, December 5). Machine Learning and The Cross-Section of Emerging Market Stock Returns. https://doi.org/10.2139/ssrn.4287550

Hanauer, M. X., & Lauterbach, J. (2018, August 17). The Cross-Section of Emerging Market Stock Returns. https://doi.org/10.2139/ssrn.3233614

Harvey, C. R., & Liu, Y. (2019, February 25). *A Census of the Factor Zoo* (SSRN Scholarly Paper No. 3341728). Social Science Research Network. Rochester, NY. https://doi.org/10.2139/ssrn.3341728

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

Ioffe, S., & Szegedy, C. (2015, March 2). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. https://doi.org/10.48550/arXiv.1502.03167

Kelly, B. T., Pruitt, S., & Su, Y. (2019). Characteristics are covariances: A unified model of risk and return. *Journal of Financial Economics*, *134*(3), 501–524. https://doi.org/10.1016/j.jfineco.2019.05.001

Lewellen, J. (2014, August 22). The Cross Section of Expected Stock Returns. https://doi.org/10.2139/ssrn.2511246

Lintner, J. (1965). The Valuation of Risk Assets and the Selection of Risky Investments in Stock Portfolios and Capital Budgets. *The Review of Economics and Statistics*, *47*(1), 13. https://doi.org/10.2307/1924119

Messmer, M. (2017, December 2). *Deep Learning and the Cross-Section of Expected Returns* (SSRN Scholarly Paper No. 3081555). Social Science Research Network. Rochester, NY. https://doi.org/10.2139/ssrn.3081555

Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill.

Moritz, B., & Zimmermann, T. (2016, March 1). *Tree-Based Conditional Portfolio Sorts: The Relation between Past and Future Stock Returns* (SSRN Scholarly Paper No. 2740751). Social Science Research Network. Rochester, NY. https://doi.org/10.2139/ssrn.2740751

Rapach, D., & Zhou, G. (2013, January 1). Forecasting Stock Returns. In G. Elliott & A. Timmermann (Eds.), *Handbook of Economic Forecasting* (pp. 328–383). Elsevier. https://doi.org/10.1016/B978-0-444-53683-9.00006-2

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

Sharpe, W. F. (1964). Capital Asset Prices: A Theory of Market Equilibrium Under Conditions of Risk*. *The Journal of Finance*, *19*(3), 425–442. https://doi.org/10.1111/j.1540-6261.1964.tb02865.x
_eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1540-6261.1964.tb02865.x

Sundararajan, M., Taly, A., & Yan, Q. (2017, June 12). Axiomatic Attribution for Deep Networks. Retrieved December 1, 2022, from http://arxiv.org/abs/1703.01365

Welch, I., & Goyal, A. (2008). A Comprehensive Look at The Empirical Performance of Equity Premium Prediction. *Review of Financial Studies*, *21*(4), 1455–1508. https://doi.org/10.1093/rfs/hhm014

Appendix A

Hyperparameters combinations for architecture search

	Type	Low	High	Choices
Activation function	Choice	-	-	ReLU, LeakyReLU
Optimizer	Choice	-	-	Adam, Nadam , Adagrad, Adadelta
L1 λ	Loguniform	1e-5	0.1	-
L2 λ	Loguniform	1e-5	0.1	-
Dropout	Uniform	0	0.7	-
Batch Normalization	Choice	0	1	-
HL1	Choice	-	-	2048,1024,512,256,64,32
HL2	Choice	-	-	2048,1024,512,256,128,64,32,16,0
HL3	Choice	-	-	1024,512,256,128,64,32,16,8,4,0
HL4	Choice	-	-	512,256,128,64,32,16,8,4,2,0

Continued on next page

	Type	Low	High	Choices
HL5	Choice	-	-	256,128,64,32,16,8,4,2,0

Table A.1: This table shows the initial search space part of the search experiment, each hyperparameter can assume one of the types allowed by the optimization algorithm implemented in the Microsoft NNI library (nni.readthedocs.io/Tutorial/SearchSpaceSpec.html), this is shown in the Type column. Hyperparameters of the type "Choice" can assume one of the values included in the column Choices, while others follow the distribution with constraints being a lower and an upper bound (Low, High).

Appendix B

Initial Firm Characteristics

Signal	Description	count	mean	std	min	max
Accruals	Accruals	2,314,866	0.0297	0.13	-11.6	12.5
AccrualsBM	Book-to-market and accruals	161,129	0.495	0.5	0	1
Activism1	Takeover vulnerability	90,795	14.8	2.76	6	23
AM	Total assets to market	2,222,621	3.54	27.1	0	1.06e+04
AnnouncementReturn	Earnings announcement return	1,944,178	0.00303	0.0933	-1.61	9.45
AssetGrowth	Asset growth	2,348,115	-0.172	3.03	-1.07e+03	1
BetaLiquidityPS	Pastor-Stambaugh liquidity beta	2,293,593	0.000653	0.473	-23.7	9.05
BetaTailRisk	Tail risk beta	1,862,743	0.608	0.482	-10.7	5.38
betaVIX	Systematic volatility	2,044,786	-0.000401	0.0178	-1.99	1.58
BMdec	Book to market using December ME	2,166,148	2.57	48.7	-4.88e+03	1.4e+04
BookLeverage	Book leverage (annual)	2,545,141	-3.54	94.2	-2.96e+04	1.19e+04
BPEBM	Leverage component of BM	2,113,036	-0.151	216	-1.21e+05	1.42e+05

Continued on next page

Signal	Description	count	mean	std	min	max
Cash	Cash to assets	1,411,778	0.139	0.188	-0.143	1
CashProd	Cash Productivity	2,190,519	19.2	4.13e+03	-1.99e+06	9.22e+05
CBOperProf	Cash-based operating profitability	1,732,452	0.101	0.199	-11.3	10
CF	Cash flow to market	2,222,621	0.0466	1.57	-560	129
cfp	Operating Cash flows to price	1,840,001	0.00509	1.49	-320	575
ChAssetTurnover	Change in Asset Turnover	1,857,118	0.312	68.6	-9.24e+03	1.51e+04
ChEQ	Growth in book equity	2,162,147	-1.29	13.3	-5.13e+03	-0.00074
ChForecastAccrual	Change in Forecast and Accrual	421,193	0.468	0.499	0	1
ChInv	Inventory Growth	2,348,139	-0.0116	0.0696	-1.71	1.69
ChInvIA	Change in capital inv (ind adj)	1,983,740	-5.87e+09	6.64e+13	-2.19e+16	7.94e+15
ChNAnalyst	Decline in Analyst Coverage	70,411	-0.119	0.324	-1	0
ChNNCOA	Change in Net Noncurrent Op Assets	2,304,812	0.0031	0.568	-25.6	166
ChNWC	Change in Net Working Capital	2,314,806	0.00547	0.474	-13.7	166
ChTax	Change in Taxes	1,973,101	0.00222	4.63	-990	3.44e+03
CompEquIss	Composite equity issuance	1,669,907	0.726	3.21	-6.46	1.95e+03
CompositeDebtIssuance	Composite debt issuance	1,434,025	-0.492	1.35	-12.3	10.4
ConvDebt	Convertible debt indicator	2,562,085	-0.146	0.353	-1	0
CoskewACX	Coskewness using daily returns	2,790,771	0.147	0.368	-3.57	6.28
CustomerMomentum	Customer momentum	198,048	0.0121	0.104	-0.981	3.33
DelBreadth	Breadth of ownership	739,794	0.132	1.14	-51.9	48.1
DelCOA	Change in current operating assets	2,348,139	-0.0254	0.124	-1.82	1.8
DelCOL	Change in current operating liabilities	2,314,866	-0.0154	0.101	-11.9	11.6
DelEqu	Change in equity to assets	2,248,695	-0.0278	0.621	-42.2	240
DelFINL	Change in financial liabilities	2,309,428	-0.0267	0.173	-12.4	28.6

Continued on next page

Signal	Description	count	mean	std	min	max
DelLTI	Change in long-term investment	2,348,139	-0.00628	0.0768	-1.94	1.8
DelNetFin	Change in net financial assets	2,309,428	-0.018	0.198	-12.4	28.6
DivSeason	Dividend seasonality	1,511,929	0.443	0.497	0	1
dNoa	change in net operating assets	2,248,671	-0.0987	2.2	-742	120
DolVol	Past trading volume	3,202,370	-1.24	3.05	-13	12.3
EarningsStreak	Earnings surprise streak	729,819	0.00358	3.61	-154	806
EarnSupBig	Earnings surprise of big firms	1,777,725	7.89e+10	5.26e+12	-6.79e+13	3.58e+14
EBM	Enterprise component of BM	2,113,036	0.656	216	-1.21e+05	1.42e+05
EntMult	Enterprise Multiple	1,840,976	-19.3	544	-2.14e+05	1.76e+03
EP	Earnings-to-Price Ratio	1,712,397	0.0899	0.118	0	22.8
EquityDuration	Equity Duration	2,207,586	-5.75e+05	1.5e+08	-4.81e+10	4.06e+06
ExclExp	Excluded Expenses	1,086,346	-0.0376	0.345	-2.59	1.65
FEPS	Analyst earnings per share	1,278,545	2.95	200	-1.3e+05	2.18e+04
fgr5yrLag	Long-term EPS forecast	616,500	-15.7	10.3	-292	212
FirmAgeMom	Firm Age - Momentum	335,494	0.107	0.373	-0.966	27.5
ForecastDispersion	EPS Forecast Dispersion	841,777	-0.222	1.54	-207	0
Frontier	Efficient frontier index	925,227	-0.0446	0.932	-11.3	21.4
Governance	Governance Index	273,753	-9.16	2.6	-14	-5
GP	gross profits / total assets	2,146,829	0.351	0.347	-31.6	12.9
GrAdExp	Growth in advertising expenses	595,425	-0.103	0.437	-7.67	4.28
grcapx	Change in capex (two years)	1,816,383	-2.08	136	-4.15e+04	9.06e+03
grcapx3y	Change in capex (three years)	1,670,678	-4.74e+10	6.34e+13	-1.44e+16	2.57e+16
Herf	Industry concentration (sales)	2,598,021	-0.315	0.285	-5.55	0
HerfBE	Industry concentration (equity)	1,898,921	-60.6	6.53e+03	-8.6e+05	0

Continued on next page

Signal	Description	count	mean	std	min	max
hire	Employment growth	2,452,612	-0.0333	0.268	-2	2
IdioRisk	Idiosyncratic risk	3,536,270	-0.0277	0.0298	-2.69	-0
IdioVol3F	Idiosyncratic risk (3 factor)	3,533,870	-0.0248	0.0265	-2.77	-0
IdioVolAHT	Idiosyncratic risk (AHT)	3,433,474	-0.0297	0.0249	-0.977	-1.02e-05
Illiquidity	Amihud's illiquidity	2,961,863	1.06e-05	0.000194	0	0.0761
IndMom	Industry Momentum	3,062,973	0.0864	0.171	-0.926	5.1
IndRetBig	Industry return of big firms	2,061,882	0.0184	0.069	-0.486	1.88
IntanBM	Intangible return using BM	1,318,480	-0.00611	0.719	-8.45	4.61
IntanCFP	Intangible return using CFtoP	1,443,884	0.00143	0.431	-40.4	25.8
IntMom	Intermediate Momentum	2,816,687	0.0773	0.451	-0.999	44.2
Investment	Investment to revenue	1,724,483	-1.01	2.08	-154	2.51e+03
InvestPPEInv	change in ppe and inv/assets	2,126,351	-0.0937	1.29	-447	13
InvGrowth	Inventory Growth	1,524,106	-0.347	25.6	-8.21e+03	1.75
LRreversal	Long-run reversal	2,391,415	-0.356	1.28	-544	1
MaxRet	Maximum return over month	3,556,907	-0.0675	0.0932	-12.4	0.87
Mom12m	Momentum (12 month)	2,834,657	0.147	0.743	-1	437
Mom12mOffSeason	Momentum without the seasonal part	2,939,467	0.0126	0.0539	-0.527	2.63
Mom6m	Momentum (6 month)	2,958,743	0.0623	0.409	-0.999	38.7
Mom6mJunk	Junk Stock Momentum	949,277	0.0651	0.433	-0.999	35.5
MomOffSeason	Off season long-term reversal	2,623,361	-0.0128	0.0249	-1.3	1.59
MomOffSeason06YrPlus	Off season reversal years 6 to 10	1,939,066	-0.0132	0.0296	-15.9	3.65
MomOffSeason16YrPlus	Off season reversal years 16 to 20	901,262	-0.0149	0.017	-0.249	0.111
MomRev	Momentum and LT Reversal	196,671	0.547	0.498	0	1
MomSeason	Return seasonality years 2 to 5	2,624,584	0.0124	0.0926	-0.906	16

Continued on next page

Signal	Description	count	mean	std	min	max
MomSeason06YrPlus	Return seasonality years 6 to 10	1,944,376	0.0127	0.082	-0.906	5.36
MomSeason11YrPlus	Return seasonality years 11 to 15	1,397,122	0.013	0.0777	-0.906	3.6
MomSeason16YrPlus	Return seasonality years 16 to 20	1,030,949	0.0133	0.0756	-0.906	3.6
MomSeasonShort	Return seasonality last year	2,838,474	0.0129	0.162	-0.984	24
MomVol	Momentum in high volume stocks	857,398	5.76	2.82	1	10
NetDebtFinance	Net debt financing	1,910,872	-0.017	0.118	-0.999	0.992
NetDebtPrice	Net debt to price	1,062,009	-1.05	5.67	-1.09e+03	186
NetEquityFinance	Net equity financing	2,330,699	-0.0212	0.111	-1	0.981
NOA	Net Operating Assets	2,253,859	-0.6	1.81	-512	270
OPLeverage	Operating leverage	2,546,240	1.02	1.15	-0.76	218
OrderBacklog	Order backlog	521,893	-0.578	1.12	-102	-9.96e-06
OrgCap	Organizational capital	927,133	-0.00616	0.975	-2.34	10.1
OScore	O Score	921,581	-0.111	0.314	-1	0
PctAcc	Percent Operating Accruals	2,157,371	1.05	58.2	-7.24e+03	1.01e+04
PctTotAcc	Percent Total Accruals	1,415,835	-2.97	173	-4.98e+04	1.97e+03
Price	Price	4,690,949	-2.48	1.27	-13.2	4.85
PriceDelayRsq	Price delay r square	3,297,711	0.387	0.325	5.56e-05	1
RD	R&D over market cap	992,971	0.0672	0.207	-0.0744	39.5
RDAbility	R&D ability	111,643	0.464	5.96	-139	64
RDcap	R&D capital-to-assets	342,369	0.149	0.524	0	19
RDIPO	IPO and no R&D spending	2,562,085	-0.00653	0.0806	-1	0
RDS	Real dirty surplus	1,829,074	-126	5.04e+03	-1.52e+06	6.92e+05
ResidualMomentum	Momentum based on FF3 residuals	2,523,224	-0.0355	0.328	-4.01	2.87
retConglomerate	Conglomerate return	398,291	0.0121	0.0792	-0.741	1.57

Continued on next page

Signal	Description	count	mean	std	min	max
ReturnSkew	Return skewness	3,493,225	-0.194	1.02	-4.9	4.9
ReturnSkew3F	Idiosyncratic skewness (3F model)	3,533,865	-0.162	0.869	-4.68	4.9
REV6	Earnings forecast revisions	250,209	-0.268	117	-5.82e+04	3.6e+03
RIO_Disp	Inst Own and Forecast Dispersion	312,765	3.61	1.22	1	5
RIO_MB	Inst Own and Market to Book	251,320	2.71	1.36	1	5
RIO_Turnover	Inst Own and Turnover	331,344	3.25	1.33	1	5
RIO_Volatility	Inst Own and Idio Vol	342,716	3.51	1.31	1	5
roaq	Return on assets (qtrly)	1,728,508	-0.00286	0.323	-58.1	172
ShareIss1Y	Share issuance (1 year)	2,704,499	-0.27	6.27	-5.67e+03	1
ShareVol	Share Volume	1,446,955	-0.284	0.451	-1	0
Size	Size	4,690,949	inf	nan	-21.8	inf
SmileSlope	Put volatility minus call volatility	429,134	-0.00213	0.133	-2.78	2.55
std_turn	Share turnover volatility	1,466,318	-0.107	1.41	-444	-6.96e-06
STreversal	Short term reversal	4,834,507	-1.01	16.8	-2.4e+03	100
SurpriseRD	Unexpected R&D increase	1,049,927	0.283	0.45	0	1
tang	Tangibility	1,153,400	0.692	0.173	0	7.15
Tax	Taxable income to income	2,276,826	1.21	19.6	-2.74e+03	4.46e+03
TotalAccruals	Total accruals	2,234,307	-0.0128	0.779	-194	120
TrendFactor	Trend Factor	1,640,554	0.233	0.126	-0.518	2.49
VolSD	Volume Variance	2,751,848	-3.29	33.1	-6.12e+03	0
XFIN	Net external financing	2,059,426	-0.0482	0.611	-54.6	166
zerotrade	Days with zero trades	3,110,857	1.74	3.55	3.38e-12	20
zerotradeAlt1	Days with zero trades	3,228,613	1.84	3.98	9.58e-13	20.2
zerotradeAlt12	Days with zero trades	2,939,846	1.66	3.35	1.38e-11	19.8

Table B.1: Initial Feature set and descriptive statistics, dataset downloaded from www.openassetpricing.com/data

Appendix C

Network architectures

	CORR	HL	LR	L1	L2	Drop	BN	OPT	ACT	ARCH	Epochs	Batch Size
Top 5 networks												
T1	7.16	3	0.1	1	0	0.7	1	Nadam	ReLU	512-256-64	200	10,000
T2	6.87	5	0.88	0.001	0.578	0.16	1	Nadam	LeakyReLU	2048-16-4-32-32	200	10,000
T3	6.79	4	0.303	0.036	1.42	0.08	0	Nadam	LeakyReLU	32-1024-32-16	200	10,000
T4	6.76	3	0.1	0.001	1	0.701	0	Adam	ReLU	512-256-64	200	10,000
T5	6.72	5	0.08	0.22	1.91	0.11	0	Adam	LeakyReLU	64-64-64-64-64	200	10,000
Bottom 5 networks												
B5	-0.72	5	0.026	0.063	0.016	0.15	0	Adadelta	LeakyReLU	32-128-128-2-64	200	10,000
B4	-1.14	5	0.166	0.012	0.384	0.42	0	Adadelta	LeakyReLU	32-128-4-16-8	200	10,000
B3	-1.31	5	0.079	0.519	0.047	0.269	0	Adagrad	ReLU	64-512-4-32-32	200	10,000
B2	-1.42	5	0.003	0.078	7.73	0.398	0	Adadelta	LeakyReLU	32-32-1024-64-128	200	10,000

Continued on next page

	CORR	HL	LR	L1	L2	Drop	BN	OPT	ACT	ARCH	Epochs	Batch Size
B1	-1.56	4	0.077	0.056	0.031	0.27	0	Adadelta	LeakyReLU	32-128-32-16	200	10,000

Table C.1: This table shows the hyperparameters of the top 5 and bottom 5 performing networks found during the optimization experiment, CORR is the out-of-sample monthly Spearman Correlation coefficient for the network in percentage points, LR is the learning rate, L1 and L2 are the λs used respectively for L1 and L2 regularization, these three are all multiplied by a factor of 1000. HL is the number of hidden layers, BN shows whether batch normalization has been used or not in the network (1 = True, 0 = False), OPT and ACT are the optimization algorithm and the activation function of the networks. ARCH shows the sequence regarding the number of neurons present in each hidden layer of the network.

Appendix D

Feature Importance

Feature	TNN Score	BNN Score
AM	0.030617	0.005569
Accruals	-0.001077	0.002260
AccrualsBM	-0.000363	0.000048
Activism1	0.000000	0.000000
AnnouncementReturn	0.009768	0.002182
AssetGrowth	0.000129	0.000234
BMdec	0.006514	0.006373
BPEBM	0.003858	-0.001114
BetaLiquidityPS	0.000000	0.000000
BetaTailRisk	0.004317	0.002853
BookLeverage	0.004632	0.001573
CBOperProf	0.002299	-0.000569
CF	0.002392	0.001026
Cash	-0.003502	0.003507

Continued on next page

Feature	TNN Score	BNN Score
CashProd	-0.001010	0.002320
ChAssetTurnover	0.000509	-0.000460
ChEQ	-0.000937	0.000079
ChForecastAccrual	0.014017	0.001866
ChInv	0.001035	-0.004103
ChInvIA	0.000664	0.000769
ChNAnalyst	0.000000	0.000000
ChNNCOA	0.000208	0.000408
ChNWC	-0.000421	0.001185
ChTax	-0.002106	-0.002090
CompEquIss	-0.002247	0.000507
CompositeDebtIssuance	0.000447	-0.001035
ConvDebt	0.004681	-0.000738
CoskewACX	0.002168	-0.000407
CustomerMomentum	0.000000	0.000000
DelBreadth	0.000000	0.000000
DelCOA	0.002344	0.000136
DelCOL	-0.000083	-0.000829
DelEqu	0.000609	-0.000104
DelFINL	0.001489	-0.000925
DelLTI	-0.000894	0.003752
DelNetFin	-0.002156	-0.000334
DivSeason	0.018426	0.002567
DolVol	0.000761	0.000788
EBM	-0.000778	0.001133
EP	0.000725	-0.001012
EarnSupBig	-0.003295	0.000622

Continued on next page

Feature	TNN Score	BNN Score
EarningsStreak	0.013251	-0.000644
EntMult	-0.001534	-0.001832
EquityDuration	-0.001754	0.000361
ExclExp	0.000042	-0.001034
FEPS	0.014324	0.001788
FirmAgeMom	-0.000001	0.000000
ForecastDispersion	0.006473	-0.003999
Frontier	0.000474	0.003150
GP	0.004839	-0.000812
Governance	0.000000	0.000000
GrAdExp	0.000694	0.000620
Herf	-0.000152	0.001629
HerfBE	0.002541	0.001934
IdioRisk	-0.004787	-0.001384
IdioVol3F	-0.008109	0.000037
IdioVolAHT	-0.009937	-0.006322
Illiquidity	0.001931	0.002676
IndMom	0.003472	0.001911
IndRetBig	-0.005858	0.000288
IntMom	-0.003589	-0.001792
IntanBM	0.005533	-0.002671
IntanCFP	-0.000363	-0.000547
InvGrowth	-0.000811	-0.000242
InvestPPEInv	0.002397	-0.000439
Investment	-0.000673	-0.000713
LRreversal	-0.001074	-0.000741
MaxRet	-0.004246	-0.004339

Continued on next page

Feature	TNN Score	BNN Score
Mom12m	0.002141	0.004424
Mom12mOffSeason	-0.002282	-0.000630
Mom6m	-0.003582	0.002914
Mom6mJunk	0.000000	0.000000
MomOffSeason	0.001607	-0.001094
MomOffSeason06YrPlus	-0.001877	-0.001513
MomOffSeason16YrPlus	0.000143	-0.002135
MomRev	0.000394	-0.000956
MomSeason	0.005180	-0.000672
MomSeason06YrPlus	0.002343	-0.000911
MomSeason11YrPlus	0.001580	0.000714
MomSeason16YrPlus	0.001532	-0.001959
MomSeasonShort	-0.001365	0.000613
MomVol	-0.002165	-0.000279
NOA	0.000475	-0.001144
NetDebtFinance	-0.002020	-0.000342
NetDebtPrice	0.001809	-0.001909
NetEquityFinance	0.001100	0.000349
OPLeverage	0.003552	0.003810
OScore	-0.010308	-0.000828
OrderBacklog	-0.000007	0.000638
OrgCap	-0.000814	0.002035
PctAcc	-0.002002	-0.001602
PctTotAcc	-0.001761	0.000915
Price	-0.004949	-0.002587
PriceDelayRsq	0.000253	-0.002463
RD	0.001461	-0.000851

Continued on next page

Feature	TNN Score	BNN Score
RDAbility	-0.001325	0.000397
RDIPO	-0.004854	0.001051
RDS	-0.000866	-0.001082
RDcap	-0.000023	-0.000116
REV6	-0.003184	-0.001748
RIO_Disp	0.000520	0.002952
RIO_MB	0.000821	0.000135
RIO_Turnover	0.003821	0.000036
RIO_Volatility	-0.000661	-0.001382
ResidualMomentum	-0.002354	-0.001399
ReturnSkew	-0.000171	-0.001104
ReturnSkew3F	0.002367	0.000037
STreversal	0.009694	0.005011
ShareIss1Y	0.001933	0.001207
ShareVol	-0.000674	-0.000077
Size	-0.002788	0.000678
SmileSlope	0.000000	0.000000
SurpriseRD	0.013724	0.001514
Tax	-0.000096	0.001636
TotalAccruals	0.001491	0.001348
TrendFactor	0.016033	0.004018
VolSD	-0.004517	0.001360
XFIN	-0.002592	-0.000907
betaVIX	0.001312	0.001248
cfp	-0.000184	0.001388
dNoa	0.000991	-0.001128
fgr5yrLag	0.007982	0.001512

Continued on next page

Feature	TNN Score	BNN Score
grcapx	-0.001697	-0.000756
grcapx3y	-0.001534	-0.000854
hire	-0.000084	-0.001817
retConglomerate	0.000000	0.000000
roaq	0.014653	0.001835
std_turn	0.001671	0.001727
tang	0.002250	0.000335
zerotrade	-0.001913	0.001788
zerotradeAlt1	-0.003835	-0.006100
zerotradeAlt12	0.006497	0.001900

Table D.1: Full list of scores derived from the Integrated Gradients method as part of the neural network model interpretation.

Declaration of Academic Integrity

Hereby, I declare that I have composed the presented paper independently on my own and without any other resources than the ones indicated. All thoughts taken directly or indirectly from external sources are properly denoted as such. This paper has neither been previously submitted to another authority nor has it been published yet.

Place, Date
Palombaro (IT), December 14$^{\text{th}}$, 2022

Signature

[illegible]