Erasmus University Rotterdam
Erasmus School of Economics
Bachelor Thesis Double Degree BSc^2 in Econometrics and Economics

Empirical Asset Pricing via Deep Learning: Machine Learning algorithms for risk premia prediction

Edoardo Cordola (537951)

Supervisor:	Terri van der Zwan
Second assessor:	Name of your second assessor
Date final version:	2nd July 2023

Abstract

This thesis performs a comparative analysis of several machine learning methods for the asset pricing goal of predicting stock risk premia. In particular, the main goal is to investigate the predictive performance of deep learning techniques with respect to traditional linear models. For this purpose US stock characteristics' data is used, for the period 1977 to 2021. Advanced deep learning models such as RNN with LSTM cells and CNN are used to perform stock excess return predictions. I find that these models achieve the highest R^2_{oos} performance among all linear and non-linear models analysed. However, long-short decile spread portfolios built on these techniques are not able to outperform the benchmark linear model in terms of Sharpe ratio. This research supports the conclusion that deep learning methods achieve their superior predictive performance thanks to their advanced ability of capturing non-linear interactions and long-term dependencies among the predictor variables.

Contents

1 Introduction

Forecasting financial risk premia has a central role in the empirical research in Asset Pricing. At the beginning of the second half of the twentieth century, Markowitz (1952) revolutionized the field of Asset Pricing with Modern Portfolio Theory. In the following years, Modern Portfolio Theory was further integrated with state-of-the-art models, such as the CAPM model (Sharpe (1964), Lintner (1965)) and the Fama-French three and five factor models (Fama and French (1993), Fama and French (2015)), which represented a disruptive discovery for the creation of factors able to describe financial returns with high explanatory power. However, this paper does not focus on the goal of further improving the explanation of the cross-section of stock returns, but rather aims at identifying efficient models able to predict future returns. Creating accurate forecasts of expected stock returns is not only crucial for asset management institutions when deciding on potential investment opportunities, but it's also fundamental in the process of creating portfolios for their clients that can efficiently leverage the excess alpha that is not captured by traditional asset pricing models. Moreover, forecasting expected returns is also extremely useful for the entire financial branch that deals with derivative products. Pricing derivative products for risk management purposes requires excellent forecasts of expected stock returns. Therefore, this research is highly relevant for institutions employing volatility and downside risks' forecast models for individual stocks or portfolios.

In this thesis the identification of relevant model architectures able to efficiently predict future excess returns is performed by applying a machine learning methodology. Specifically, considerable attention is devoted to the application of non-linear model specifications and the comparison between their predictive performance and the one of traditional (linear) empirical methods in asset pricing. Given their extensive flexibility in dealing with high dimensional data and in creating powerful model specifications, machine learning methods are indeed extremely well suited to the matter of predicting expected stock returns. This is also the case why the interest in the application of these statistical models to financial matters has grown substantially in recent years. However, the enhanced flexibility of machine learning models also makes them prone to the risk of overfitting the data. This risk is addressed by applying regularization techniques and by restricting the forecasts' evaluation to their out-of-sample performance, making our analysis robust against this threat.

The research of Gu et al. (2020) focuses on a wide selection of machine learning methods, identifying neural networks among the models with the best predictive performance of future returns. However, besides introducing Feedforward Neural Networks (FNN), Gu et al. (2020) place limited emphasis on more advanced deep learning methods. In this paper I replicate most of the machine learning models from the research of Gu et al. (2020), but I also implement three additional deep learning models: Recurrent Neural Networks (RNN) with LSTM (Hochreiter and Schmidhuber (1997)) and GRU unit layers (Cho et al. (2014)) and Convolutional Neural Networks (CNN) (LeCun (1989)). Moreover, I also extend the analysis of Gu et al. (2020) in the context of portfolio creation, leveraging the forecasts obtained for individual stock returns. In particular, I analyse the relative performance of the aforementioned deep learning techniques in

delivering risk-adjusted portfolio returns with respect to less complex linear models, such as the benchmark OLS. Value-weighted and equal-weighted portfolios are created using a long-short decile spread strategy and the results are compared against the benchmark OLS.

Therefore, this research will focus on the evaluation of the potential gains of deep learning algorithms in predicting excess stock returns. In particular, the central research question is:

To what extent do deep learning algorithms such as RNN and CNN improve the predictive performance of excess stock returns compared to traditional empirical asset pricing models?

Using data from CRSP for the period, I find that these models achieve the highest R^2_{oos} performance among all linear and non-linear models analysed. However, long-short decile spread portfolios built on these techniques are not able to outperform the benchmark linear model in terms of Sharpe ratio.

The remainder of this thesis is organized as follows. Section 2 describes the main literature from the field. Section 3 describes the data and Section 4 discusses the methodology, while Section 5 describes the main results. Finally, in Section 6 I discuss the conclusions of the analysis.

2 Literature

2.1 Forecasting excess stock returns

Our research builds upon the existing literature that identifies relevant predictors of future stock returns. In the second half of the twentieth century, Fama (1970) presented the "Efficient Market Hypothesis", stating that all available information in the market is already reflected in the asset prices. Therefore, according to this hypothesis, predicting future stock returns relying only on present information was unfeasible, as any potential future fluctuation being already priced in. However, in later years Rosenberg et al. (1985) challenged this theory, providing evidence for better performance of actively managed funds and, therefore, leaving open the debate concerning the effectiveness of future returns predictability. Fama and French (1988) identified dividend yields as a driving variable of future expected stock returns. Campbell and Shiller (1988a) extended on this hypothesis by analysing the impact of dividends and expectations of future dividends on expected returns. In particular, they provided evidence of the linear forecasting relationship between the dividend-price ratio and future stock returns. Moreover, in a subsequent paper, Campbell and Shiller (1988b) extended on their previous research, by analysing the existent correlation between changes in expected future dividends and future expected returns, offering additional insights into the predictability of stock returns. Finally, Jegadeesh and Titman (1993) provided evidence of the short-term return predictability power of momentum, by constructing trading strategies based on this variable.

At the beginning of the twenty-first century, Cochrane (2008) once again rejects the Efficient Market Hypothesis, emphasising the presence of some degree of asset returns' predictability.

Particularly, several factors that exhibit predictive power for future returns are found, such as valuation ratios, dividend yields and macroeconomic variables. On the other hand, Welch and Goyal (2008) performs time series regressions of stock returns on a wide collection of firm characteristics that are suspected to drive risk premiums. Welch and Goyal (2008) present not so encouraging results, not identifying any relevant firm characteristic able to provide meaningful and robust empirical forecasting power. Moreover, Campbell and Thompson (2008) show that while certain predictors may demonstrate some predictive power, a simple strategy relying on historical averages is hardly beaten in an out-of-sample estimation. Finally, Rapach et al. (2010) explore the combination of mutliple forecasts to deal with this problem, achieving statistically and economically significant out-of-sample gains relative to the historical average over time.

2.2 Machine learning for forecasting excess stock returns

In recent times the popularity of machine learning techniques in the financial field has grown substantially. In particular, given the frictions encountered in the application of traditional linear models for asset returns' predictions, the asset pricing field has witnessed increasing applications of machine learning techniques. Specifically, recent literature of this new stream of research aims at modeling non-linear dependencies between assets' characteristics and returns, leveraging the intrinsic propensity of machine learning methods to deal with large amount of variables and their effectiveness in dealing with low signal-to-noise environments, such as the financial market. In this context, Gu et al. (2020) analyse a wide variety of machine learning methods to predict stock returns and construct sorted portfolios. In particular, their findings demonstrate that deep learning techniques, such as feedforward neural networks, exhibit superior performance compared to all benchmarks. This superiority is observed in terms of prediction accuracy as well as the Sharpe ratio of prediction-sorted portfolios. Feng et al. (2018) are among the first to apply machine learning algorithms to the prediction of asset returns. In particular, they find the existence of nonlinear factors which explain predictability of returns.

An extensive part of the literature in deep learning is not only limited to feedforward neural networks, as Gu et al. (2020) mainly focus on, but also explores the application of more advanced techniques to financial data. Chen et al. (2023) introduce a generative adversarial network (GAN) consisting of a recurrent neural network (RNN) with long short-term memory (LSTM) cells to estimate a conditional stochastic discount factor (SDF) without specifying its functional form. LSTM cells are used to model long-term macroeconomic dependencies to then be combined with firm characteristics in a standard FNN. On the other hand, the GAN is a powerful tool to identify portfolios with the most unexplained pricing information, such that the SDF can be tuned accordingly. Another innovative paper in this field is by Cong et al. (2021), who create a framework to construct optimal portfolios by using deep reinforcement learning. Once again, in this framework LSTM is used to extract information and long-term dependencies from the input features, such as firm fundamentals. These "Alphaportfolio" models yield extremely high out-of-sample performance.

Other streams of research in the application of machine learning techniques to financial purposes are by Gu et al. (2021), who use a deep learning autoencoder to create latent factor conditional asset pricing model, and by B. T. Kelly et al. (2020), who use Principal Component Analysis to create another latent factor asset pricing model.

3 Data

The data consists of 92 firm characteristics and associated monthly simple returns obtained from CRSP from January 1977 to December 2021. Furthermore, one-month and annual Treasury bill yields are retrieved from the Kenneth R. French Data Library and used as a proxy for the risk-free rate. This monthly and yearly observations are subtracted from the simple returns in order to obtain excess returns, which is used as target variable in the analysis. Moreover, I use standard filters to include only stocks listed on the NYSE, AMEX, or NASDAQ for more than one year, and to exclude stocks with negative book equity or lag market equity.

Furthermore, as most of these figures are released to the public with a delay, it is important to carefully deal with a potential forward-looking bias. Specifically, we assume that monthly characteristics are delayed by at most a month, quarterly with at least 4 months lag, and annually with at least 6 months lag. Hence, in order to predict returns at month $t+1$, we use most recent monthly characteristics at the end of month t, most recent quarterly data by end $t-4$, and most recent annual data by end $t-6$.

Missing characteristics are imputed using the local B-XS model of Bryzgalova et al. (2022). Moroever, all stock characteristics are cross-sectionally standardized period-by-period in the interval [-1, 1], following B. T. Kelly et al. (2019). This is called a rank transformation. More specifically, the following formula is applied for each characteristic at a particular point in time:

$$z_x = \frac{rank_x - 1}{n - 1} \times 2 - 1,$$

where $rank_x$ is the rank of the value x in the column, the smallest value has rank 1 and the largest value has rank n, with n being the total number of values in the column. This standardization is performed because of its insensitivity to outliers and the improved model performance and computational time that results from it.

In the analysis the data is split into 3 sets. The training (in-sample) set ranges from 1977/01/31 to 1990/12/31, the validation (in-sample) set from 1991/01/31 to 1999/12/31 and the testing (out-sample) set ranges from 2000/01/31 to 2021/12/31. All models requiring hyperparameter tuning are trained using the training and validation set, while for methods with no need for validation the two sets are merged together. Therefore, the actual evaluation of the models is performed out-of-sample, recreating a realistic environment in which future stock returns are of course unknown. Moreover, when forecasting out-of-sample on the testing set, the model is re-fit every year (instead of every month) by means of an expanding window that enlarges the training set while keeping the validation set fixed.

The analysis is performed using both excess monthly and annual excess returns. To construct annual returns several transformations are performed. First, log returns are computed as $r_t = \ln(1 + R_t)$, with R_t being simple monthly returns. Second, at every point in time, for every firm that exhibits monthly returns for the past 12 months (current one included), the annual log return is computed as the sum of the last 12 monthly log returns. Firms that miss at least one observation in that time frame are dropped from the data set. Finally, annual continuously compounded returns are transformed back into simple annual returns by taking $R_t = \exp(r_t) - 1$. The risk-free rate (annual Treasury Bill yield) is then subtracted from the simple annual returns. Therefore, when fitting the models to annual data, the training and validation sets are shifted by a year, keeping their initial length of respectively 14 and 9 years and shortening the testing set by one year.

4 Methodology

The goal of this thesis is to assess whether machine learning algorithms improve the predictive performance of expected stock returns compared to traditional linear models. This translates to the problem of estimating stock excess returns r as a function f of some predictor variables X and a vector of model parameters θ. Specifically, the estimation of stock's excess returns can be framed as an additive prediction error model of the form:

$$r_{i,t+1} = E_t(r_{i,t+1}) + \varepsilon_{i,t+1},$$

where

$$E_t(r_{i,t+1}) = f(x_{i,t}; \theta).$$

Stocks are in this case indexed as $i = 1, ..., N_t$ and months by $t = 1, ..., T$. Our goal is to estimate several linear and non-linear specifications for the function f, maximizing the out-of-sample explanatory power for realized excess returns $r_{i,t+1}$. It is important to mention that the function $f(\cdot)$ maintains the same form over time and across different stocks, such that the model employs information from the entire panel, allowing for a stable estimation of stock's risk premia.

4.1 Linear Regression (OLS)

Our analysis starts with a Linear Regression model estimated using ordinary least squares (OLS). Excess returns are modeled as a linear combination of stocks' features and a parameter vector, θ,

$$f(x_{i,t}; \theta) = x'_{i,t}\theta.$$

This model estimates the parameters θ by means of the standard least squares loss function:

$$\mathcal{L}(\theta) = \frac{1}{NT} \sum_{i=1}^{N} \sum_{t=1}^{T} (r_{i,t+1} - f(x_{i,t}; \theta))^2$$

By minimizing this loss function we are able to estimate the pooled OLS estimator that can be expressed in the closed form solution $\hat{\theta} = (X'X)^{-1}X'r$. Of course, this closed form analytical solution presents some advantages in terms of computational time with respect to other machine learning method.

4.2 Linear Regression with Huber Loss function

One of the key assumptions of Ordinary Least Squares estimation lies in the homoskedasticity of the shocks' variances. However, if this is not the case, it is functional to estimate a linear regression model with a weighted least squares objective function, such as:

$$\mathcal{L}(\theta) = \frac{1}{NT}\sum_{i=1}^{N}\sum_{t=1}^{T} w_{i,t}(r_{i,t+1} - f(x_{i,t};\theta))^2$$

This loss function is then effective in weighing different observations according to their statistical or economical significance. In particular, a higher weight is generally given to observation with smaller variance, given their higher degree of information.

In the financial literature it is common knowledge that assets' returns exhibit some "stylized facts", that need to be taken into account in order to perform robust analyses. In particular, the distribution of financial returns is known to be non normal, with heavy tails and slightly negatively skewed. Since OLS forecasting power is extremely sensitive to outliers, the Huber robust objective function is implemented the training set in order to counteract the misleading effect of fat-tailed observations. The Huber Loss, $H(\cdot)$, presents the usual squared loss for relatively small errors, while for large errors it employs an absolute value loss scaled by the error term δ and corrected by a term δ^2. The Huber loss function takes the following form:

$$\mathcal{L}(\theta) = \frac{1}{NT}\sum_{i=1}^{N}\sum_{t=1}^{T} H(r_{i,t+1} - f(x_{i,t};\theta), \delta),$$

where

$$H(x,\delta) = \begin{cases} x^2, & \text{if } |x| \leq \delta \\ 2\delta|x| - \delta^2, & \text{if } |x| > \delta \end{cases}$$

In our analysis, the parameter δ is set to be equal to the maximum between the 99.9 percentile of the absolute residual and 1, in order to save computational time:

$$\delta = \max\left(|r_{i,t+1} - f(x_{i,t};\theta)|_{0.999}, 1\right).$$

4.3 Penalized Linear: Elastic Net

When predicting risk premia using a large amount of model parameters, the linear regression model can exhibit overfitting, especially considering notorious low signal-to-noise ratio of stock returns. In order to tackle this issue, regularization techniques are a popular approach in the machine learning literature, given their ability to deteriorate in-sample performance to improve

the out-sample one. In particular, a penalty term is added to the original loss function:

$$\mathcal{L}(\theta;\cdot) = \frac{1}{NT}\sum_{i=1}^{N}\sum_{t=1}^{T}(r_{i,t+1} - f(x_{i,t};\theta))^2 + \phi(\theta;\cdot)$$

Particularly, we use the "elastic net" penalty, taking the following form:

$$\phi(\theta;\lambda,\rho) = \lambda(1-\rho)\sum_{j=1}^{P}|\theta_j| + \frac{1}{2}\lambda\rho\sum_{j=1}^{P}\theta_j^2$$

The elastic net involves two non-negative hyperparameters, λ and ρ, which can be tuned to achieve two specific regularizers. The first is obtained when setting $\rho = 0$, corresponding to the "Lasso" (Tibshirani (1996)) or "L1" parameter penalization. When this regularization is selected, the absolute value operator is applied to the paramaters in the objective function, allowing the model to set coefficients of a subset of features exactly equal to zero. In this manner, sparsity in the coefficients is ensured, introducing a principle of variable selection in the model. On the other hand, when setting $\rho = 1$, the "Ridge" (Hoerl and Kennard (1970)) or "L2" regularization is introduced. In this specification, the parameters in the loss function are squared; hence, when minimizing the loss function all the coefficients are shrunk towards zero. This helps avoiding parameters unnecessary large in magnitude, introducing a shrinkage principle in the estimation. Therefore, elastic net incorporates a linear combination of these two regularization, allowing for both coefficient shrinkage and feature selection. The parameters λ and ρ are tuned in the validation sample by means of coordinate descent.

4.4 Dimension reduction: PCR and PLS

As explained by Gu et al. (2020), forming linear combinations of predictors helps reduce noise, isolating better the signal in the features, and helps decorrelate highly inter-related predictors. In this thesis we analyse Principal Components Regression (PCR) and Partial Least Squares (PLS), two prominent dimension reduction techniques.

These two techniques differ for their ability to incorporate the statistical goal of forecasting returns in the dimension reduction step. In particular, while PCR fails to incorporate the forecasting objective when selecting a set of linear combinations of components that retain most of the common variation in the data, PLS performs dimension reduction by selecting the predictors that experience the highest partial sensitivity of returns.

PCR and PLS are implemented by reorganizing the linear regression equation into matrix form as follows:

$$R = X\theta + E,$$

where R is the $N \times 1$ vector of $r_{i,t+1}$, X is the $N \times P$ matrix of stacked predictors $x_{i,t}$, and E is a $N \times 1$ vector of residuals $\varepsilon_{i,t+1}$.

Both PCR and PLS aim at reducing the number of predictors from P to a much smaller number of K linear combinations of them. Ultimately, the forecasting model can be expressed as follows:

$$R = (X\Omega_K)\theta_K + \tilde{E},$$

where Ω_K is a $P \times K$ matrix with columns $w_1, w_2, \ldots, w_K$ that represent the weights given to each row of X for the creation of j_{th} principal component. Therefore, the difference between PCR and PLS lies in the choice of these weights, that in the former solve the following optimization problem:

$$w_j = \arg\max_w \text{Var}(Xw), \text{ s.t. } w'w = 1, \text{ Cov}(Zw, Zw_l) = 0, l = 1, 2, \ldots, j-1,$$

while for the latter:

$$w_j = \arg\max_w \text{Cov}^2(R, Xw), \text{ s.t. } w'w = 1, \text{ Cov}(Zw, Zw_l) = 0, l = 1, 2, \ldots, j-1$$

4.5 Regression Trees

As opposed to linear regression models, regression trees achieve the goal of modelling non-linear interactions between the covariates and the dependent variable. Specifically, the tree splits the data into several groups of observations and within each partition the average value of the outcome variable is retained. The procedure to construct a regression tree is sequential and starts with the choice of a maximum depth of the tree. At every step the groups of data obtained from the previous iteration are further split into branches according to a threshold value of a predictor variable. In Figure 1 a graphical representation of a regression tree is reported. In this instance, a tree of maximum depth equal to three is considered, with the data being split into several partitions based only on two features, size and value. At every node, the data is sorted into two leaf nodes based on a specific feature and a numerical threshold. Finally, the predicted return for an observation that lies in one of the end nodes is equalled to the average return of the observations in that specific node. At every node of the tree a decision is to be made on the split criterion to use. In particular, the algorithm of Breiman et al. (1984) is used in order to select features and the threshold to split on. This algorithm at each iteration greedily selects a splitting criterion such that the forecast error is minimized.

However, the great flexibility of regression trees to model non-linear patterns in the features exposes them to the risk of overfit. To avoid this scenario, regularization techniques are crucial in order to achieve a robust estimation out-of-sample. In our comparative analysis we explore two prominent regularization techniques, namely "Bagging" and "Boosting".

4.5.1 Random Forest

Bagging (Breiman (1996)), also known as booststrap aggregation, is a prominent regularization technique in which B different bootstrap samples of size N_b are drawn from the data. To every bootstrap sample is fit a tree and the final prediction is computed as the average of all the individual tree predictions. We implement Random Forest (RF) (Breiman (2001)), a variation

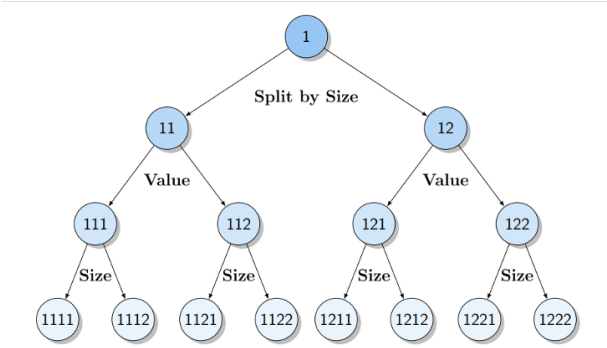

Figure 1: Example of a Regression Tree based on size and value. Figure retrieved from Bryzgalova et al., 2021.

of bagging that ensures a reduction of correlation among the trees of different bootstrap samples. In particular, this is achieved by the "dropout" feature, which regulates the number of features used to select the splitting criterion. This helps building regression trees with a certain degree of feature variation in low-level splits, reducing the correlation between the trees of different bootstrap samples.

4.5.2 Gradient Boosting

Tree Boosting (Freund et al. (1999)) is a machine learning algorithm that combines multiple shallow decision trees into a single strong predictive model. At each step, an additional tree of the same depth is fitted to the prediction residual of the previous ones. The forecasts of the ensemble of trees are combined together, but the forecasts created from past residuals are shrunk by a factor $\nu \in (0, 1)$, referred to as learning rate. This is done in order prevent overfitting of the prediction residuals. This iterative algorithm is stopped when a group of B trees is generated. Gradient boosting is a variant of boosting that employs the gradients of the loss function in order to train the creation of boosting trees. In our case, we employ the regularization technique of Extreme Gradient Boosting (XGBoost) by Chen and Guestrin (2016), which makes use of a weighted-quantile and sparsity-aware split finding that results to be computationally more efficient. Moreover, we also implement Light Gradient Boosting Machine (LGBM), which differs from Extreme Gradient Boosting in the tree growth strategy. While XGBoost grows trees level-by-level, splitting all nodes at each step, LGBM uses a leaf-wise strategy, focusing on splitting the leafs that generate the maximum loss reduction. Since LGBM focuses on the most informative leafs, this usually implies a faster convergence rate, better model performance but also more risk of overfitting. On the other hand, the level-wise growth strategy of XGBoost is more memory-intensive and therefore slower in convergence, providing a more balanced tree structure that is less prone to overfitting but also less able to capture fine-grained patterns in the data.

4.6 Neural Networks

The focus of the aforementioned models is to capture linear and simple non-linear interactions (using splitting) between features and excess returns. Nevertheless, considering the intricate structure of financial markets' dynamics, highly complex and flexible non-linear model

specifications are desirable in order to best capture the underlying relationship between stock characteristics and excess returns. Neural Networks represent one of the pillars of machine learning methodology, given their extensive aptitude to model highly complex non-linear functions. Hornik et al. (1989) proved that neural networks are theoretically able to model any continuous function to an arbitrary level of precision, identifying them as "universal approximators" for any smooth predictive association.

4.6.1 Feedforward Neural Networks

The first model architecture that we analyse is the traditional "feed-forward" network. Analogously to the transfer of information in the human brain by means of neurons and synapses, Feedforward Neural Networks (FNN) consists of a collection of nodes (neurons) that are organized into layers. Each node is connected to every other node in previous and subsequent layers. More specifically, a FNN is composed by an input layer, L hidden layers with l_i nodes in each layer i and a final output layer that aggregates the information into a final outcome prediction. The overall number of hidden layers is referred as the depth of the model, from which follows the connotation "Deep Learning".

Considering the model architecture, every hidden layer is defined as a function of the layer that preceded it. Hence the j_{th} layer is identified as:

$$h_j = \sigma(W_j h_{j-1} + b_j),$$

with $j > 1$ and where the parameters are can be interpreted as weights and biases. We consider FNN with up to 5 hidden layers, where the number of neurons in each layer is selected according to the geometric pyramid rule. Hence, if for instance we are dealing with a network with 5 hidden layers, we would have layers with 32, 16, 8, 4, 2 nodes, respectively, while in case of only 1 hidden layer, 32 neurons will be the standard set-up. Therefore, we construct a FNN with the following specification:

$$f(x_t;\theta) = W_{\text{out}}\sigma(W_L\sigma(W_{L-1}...\sigma(W_1 x_t + b_1)... + b_{L-1}) + b_L) + b_{\text{out}}$$

where $\sigma(x) = \max(x, 0)$ is the Rectified Unit (ReLU) activation function. This activation function is particularly popular due to its ability in mitigating the vanishing gradient problem and in encouraging sparsity in the number of active neurons.
The parameters $\theta = [b_1, \dots, b_L, b_{\text{out}}, W_1, \dots, W_L, W_{\text{out}}]$ are estimated minimizing the following loss function, where it is included a "Lasso" L1 regularization term:

$$\mathcal{L}(\theta) = \frac{1}{NT}\sum_{i=1}^{N}\sum_{t=1}^{T}(r_{i,t+1} - f(x_{i,t};\theta))^2 + \lambda\sum_{j=1}^{P}|\theta_j|$$

To minimize the loss function and train the neural network, a variant of Stochastic Gradient Descent (SDG) that employs backpropagation and learning rate shrinkage is used, namely the ADAM optimizer (Kingma and Ba (2014)). The learning rate, which controls the step size of the descent, is shrunk as the gradient approaches zero, such that the problem of vanishing or

exploding gradients is mitigated. Moreover, instead of a "Ridge" L2 penalty term in the loss function, an "Early Stopping" regularization is implemented. This implies that the training procedure is halted when the validation performance starts to decrease, preventing the model from overfitting. Furthermore, "Batch Normalization" (Ioffe and Szegedy (2015)) is used to reduce the phenomenon of internal covariance shift that generates from the variability of predictors across different regions of the network. By normalizing each hidden layer input in each training step (a "batch"), this algorithm enhances the performance of the neural network, allowing for higher learning rates and relying less on the initial parameter initialization. Finally, we make use of a Dropout regularization, which works by randomly setting a fraction of the neurons to zero during each training iteration. This helps introducing noise and variability during the training process, making the neural network more robust, with improved generalization performance and therefore less prone to overfitting. In order to strike a balance between preventing overfitting in the early stages of the training and allowing the model to capture more intricate patters, the dropout rate is set to gradually decline across epochs.

4.6.2 Recurrent Neural Networks: LSTM and GRU

Recurrent Neural Networks (RNN) are a variety of neural networks designed to process sequential data by means of feedback loops. This allows the information to persist in several layers, making it possible to model long-term relationships, which are an intrinsic characteristic of financial data. In this research, particular attention is placed on gated RNNs. This type of networks are developed on the idea of creating paths through time that have derivatives that neither vanish nor explode. Besides being able to accumulate information over time, gated RNNs present the fundamental capacity of learning when the information accumulated is not anymore useful to the network, clearing the old state of the cell when this is the case. In this research we focus on two types of gated RNNs, namely "Long Short-Term Memory" (LSTM) and "Gated Recurrent Unit" (GRU).

The long short-term memory (LSTM) model of Hochreiter and Schmidhuber (1997) represents the first introduction of self-loops that don't cause the problem of vanishing or exploding gradient inside a RNN. In contrast to the classic hidden units of RNNs, LSTM model presents a system of gating units that controls the flow of information. Specifically, an LSTM unit consists of a cell, which performs the crucial mansion of managing the memory of the unit by means of a linear self-loop, an *input gate* a *forget gate* and an *output gate*. At each step a new memory cell $\tilde{c}_t$ is created taking the input x_t and the previous hidden state h_{t-1}:

$$\tilde{c}_t = \tanh\left(W_h^c h_{t-1} + W_x^c x_t + w_c^0\right).$$

The input i_t, output o_t and forget f_t gate are computed as follows:

$$\begin{aligned} i_t &= \sigma\left(W_h^i h_{t-1} + W_x^i x_t + w_0^i\right), \\ f_t &= \sigma\left(W_h^f h_{t-1} + W_x^f x_t + w_0^f\right), \\ o_t &= \sigma\left(W_h^o h_{t-1} + W_x^o x_t + w_0^0\right), \end{aligned}$$

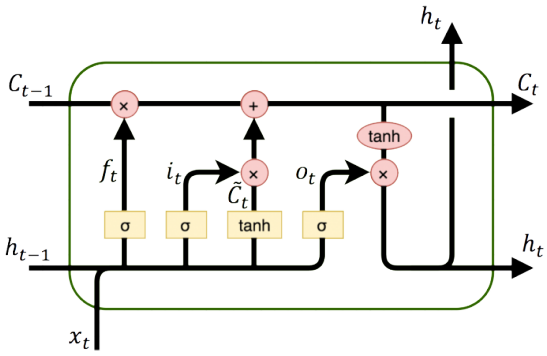

Figure 2: Illustration of the components of a LSTM unit.

where σ represents the sigmoid function, which outputs a value between 0 and 1 and where W_h, W_x, w_0 are respectively the recurrent weights, input weights and biases associated to each type of gate. The final memory cell c_t and hidden state are then computed as:

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t,$$

$$h_t = o_t \odot \tanh(c_t),$$

where $\odot$ denotes element-wise multiplication. Long-term dependencies are stored in the hidden state h_t and are used in the next layer. An overview of the structure of a LSTM unit is given in Figure 2.

Gated Recurrent Units or GRUs (Cho et al. (2014); Chung et al. (2014); Chung et al. (2015); Jozefowicz et al. (2015)) differ from LSTM units in the fact the they combine the memory cell and the hidden state into a single vector, simplifying the gating system. Since it employs less parameters than LSTM, it is computationally more efficient, while still being effective in capturing long-term dependencies. For more details on GRU and RNNs in general, we refer to Goodfellow et al. (2016).

The model architecture, loss function and regularization techniques used to build these two types of RNNs are similar to the ones employed for FNNs. Therefore, the main difference from the architecture of FNNs is the introduction of layers of LSTM or GRU units, respectively, between the input and the hidden layers.

4.6.3 Convolutional Neural Networks

Convolutional Neural Networks (CNN) (LeCun, 1989) are another variety of neural networks, popular in the field of image processing. As stated by Goodfellow et al. (2016), *convolutional networks are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers.* CNNs can be applied to financial time series by employing a 1-dimensional convolution over the sequence of data. CNNs consist of convolutional layers, pooling layers and kernels. Convolutional layers are used to extract features in the data by means of a set of filters, also known as kernels. These kernels are used to perform the convolutional operation

on the input data, identifying local patterns and features. The feature map is the output produced from the convolutional operation and represents the activation of the learned features at different locations of the input. Pooling layers are then introduced inside the network to reduce the spatial dimension of the feature map, by aggregating neighbor values and retaining the most relevant information. Moreover, CNNs are particularly well suited to our goal of predicting excess returns using stock characteristics given their translation invariance property. More specifically, pooling layers operate by extracting the presence of the most meaningful features, regardless of their precise location. The translation invariance property is therefore particularly useful for time series that exhibit time-varying patterns, making the output of the convolutional operation robust to stochastic features' patterns. Once again, the architecture, loss function and regularization techniques are similar to those employed for FNNs. However, convolutional and pooling layers are introduced between the input and the hidden layers. For more details on Convolutional Neural Networks, we refer again to Goodfellow et al. (2016).

4.7 Performance Evaluation

The predictive performance of individual excess stock returns forecasts is assessed by means of the out-of-sample R^2:

$$R^2_{oos} = 1 - \frac{\sum_{(i,t)\in\tau_3}(r_{i,t+1} - \hat{r}_{i,t+1})^2}{\sum_{(i,t)\in\tau_3} r^2_{i,t}},$$

where τ_3 indicates that the performance is computed only out-of-sample on the testing set. It is important to notice that in the denominator the sum of squared excess returns is computed without demeaning. This is the case because for individual stock returns using historical averages as benchmark typically underperforms a naive forecast of zero substantially. This implies that the historical average of stock returns is generally characterised by consistent noise, which lowers the bar for good forecasting performance. Therefore, the R^2 metrics is benchmarked against a base forecast value of zero, making the analysis findings more robust and hence more meaningful.

Pairwise comparison of out-of-sample performance between models is made by means of the Diebold and Mariano (1995) test statistic. In particular, I employ the Diebold-Mariano test statistics developed by Gu et al. (2020), which compares the cross-sectional average of prediction errors from each model, instead of comparing errors among individual returns. In particular, the following test statistic is defined:

$$DM_{12} = \frac{\bar{d}_{12}}{\hat{\sigma}_{d_{12}}},$$

where

$$d_{12,t+1} = \frac{1}{n_{3,t+1}} \sum_{i=1}^{n_{3,t+1}} ((\hat{e}^{(1)}_{i,t+1})^2 - (\hat{e}^{(2)}_{i,t+1})^2),$$

$\hat{e}^{(1)}_{i,t+1}$ and $\hat{e}^{(2)}_{i,t+1}$ denote the prediction error for stock i at time t using each method, and $n_{3,t+1}$ is the number of stocks in the testing sample, namely at year $t+1$. Finally, $\bar{d}_{12}$ and $\hat{\sigma}_{d_{12}}$ denote the mean and the Newey-West standard error of $d_{12,t+1}$ over the testing sample. Newey-West standard errors are employed given their robustness with respect to the presence of autocor-

relation in the data. Byusing this regularizations, the Diebold-Mariano test statistics is more likely to return appropriate p-values. The Diebold-Mariano test statistics are standard normally distributed $\mathcal{N}(0,1)$ under the null hypothesis of no difference between the two models.

Moreover, in order to identify the stock features that have a major influence on the cross-section of stock returns,a variable importance measure is created for every covariate within each model. In particular, I employ the variable importance measure VI_j of B. Kelly et al. (2019), which is defined as the reduction in panel predictive R^2 from setting all values of the j_{th} variable to zero while holding the other covariates fixed. This estimation is performed using the (in-sample) training set, and for each variable the mean VI_j estimated after refitting the model each year is retained.

4.8 Portfolio Construction

To further assess the predictive performance of the individual return predictions, machine learning portfolios are created based on them. In particular, I construct portfolios by means of a long-short strategy that at the end of each month sorts stocks into deciles, based on the each model's predictions. A zero-net investment strategy is then performed, by buying the stocks with the highest expected return (decile 10) and shorting the stocks with the lowest expected return (decile 1). To do so, all the models are fitted using as target variable simple returns (without subtracting the monthly Treasury bill yield), with their predictive performance being reported in the Appendix. The stocks are aggregated into portfolios using value-weight, with the largest stocks by market capitalization earning bigger weights, and an equal-weight strategy.

In order to evaluate the performance of each portfolio, we employ several metrics. Given the protfolio returns, average return and volatility are calculated by taking the mean and standard deviation, respectively, and then annualized by multiplying by the square root of the number of months in a year. The Sharpe ratio is computed by simply taking the ratio between mean returns and volatility. Risk-free rate is neglected in the Sharpe ratio computation, as the main goal here being to compare relative performance across all models.

5 Results

5.1 Individual forecasts performance

Table 1 presents the comparison of all machine larning methods in terms of their out-of-sample predictive R^2_{oos}. I compare a total of fifteen models, including OLS with all covariates, OLS-3 (which preselects size, book-to-market, and momentum as the only covariates), PLS, PCR, elastic net (ENet), random forest (RF), light and extreme gradient boosting regression trees (LGBM, XGB), feedforward neural network architectures with one to five layers (NN1,...,NN5), recurrent neural networks with LSTM and GRU units and convolutional neural networks (CNN). For OLS I perform also the robust version with Huber loss function.

Table 1: Monthly out-of-sample prediction performance (percentage R^2_{oos})

	OLS +H	OLS-3 +H	PLS	PCR	ENet	RF	LGBM	XGB	FNN (NN5)	LSTM	GRU	CNN
All	-0.20	-0.14	0.35	0.32	0.38	0.02	0.31	-0.72	0.38	0.52	0.49	0.43
Top 1000	-0.16	-0.08	0.25	0.32	0.35	0.01	0.55	-1.39	0.47	0.76	0.70	0.65
Bottom 1000	-0.60	-0.51	0.42	0.36	0.42	0.11	0.30	-0.29	0.39	0.47	0.44	0.39

Note. +H indicates the use of a Huber loss. For Feed Forward Neural Networks the best model is reported. Estimation ranges from 1 to 5 hidden layers, with NN5 being the most performing model.

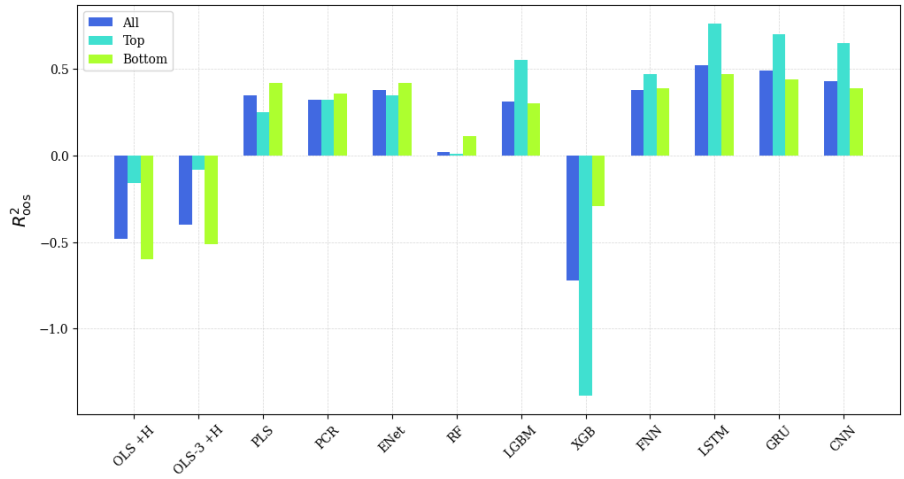

Figure 3: Visual comparison of the R^2_{oos} statistics from Table 1

The first row of Table 1 reports the R^2_{oos} for the entire sample. All the models are trained using all 92 stock features. Both the OLS and OLS-3 models, which are trained using the Huber loss function, perform slightly worse than a naive forecast of zero to all stocks in every month, with an R^2_{oos} of -0.20% and -0.14%, respectively. Therefore, restricting the amount of covariates to size, book-to-market and momentum improves the out-of-sample performance of the model minimally. Imposing some regularization techniques, such as dimension reduction, improves the accuracy of the predictions substantially. When forming linear combinations of predictors via PLS and PCR, the R^2_{oos} improves remarkably, achieving values of 0.35% and 0.32%, respectively. The rationale behind this improvement is that stock characteristics are notoriously noisy and most of the times also redundant. Reducing the dimension of the feature space and combining the characteristics into principal components helps in revealing the correlated signals, diminishing their intrinsic noise. Another crucial regularization procedure is obtained by applying ENet. In the model specification used in the analysis, the parameter ρ is set equal to 0.5, such that it performs both feature selection ("Lasso") and coefficient shrinkage ("Ridge"). When applying this specification to the data, even better out-of-sample performance is achieved, with an R^2_{oos} of 0.38%. This further emphasizes the importance of variable selection and coefficient shrinkage when dealing with an environment characterised by a fundamental low signal-to-noise ratio and high multicollinearity.

When analysing regression trees, the performance of Light Gradient Boosting Machine (LGBM) stands out with respect to Extreme Gradient Boosting (XGB) and to random forest. LGBM provides an R^2_{oos} of 0.31%, which outperforms both the 0.02% of RF and the -0.72% of XGB. The reason behind this surprising outcome could be identified by two factors. First, the better propensity of LGBM compared to XGB to find model performance thanks to its leaf-wise growth strategy, which focuses only on the most informative leafs. Second, both RF and XGB grow trees level-by-level, making the training process more balanced but at the same time more lenghty and less likely to identify fine-grained patterns in the data. Both RF and XGB were tuned to estimate relatively shallow trees, given the lower computational cost.

Neural network methods are the best performing nonlinear method and best predictor overall. In particular, for FNN the R^2_{oos} peaks for the model architecture with five hidden layers, with a value of 0.38%. However, it's worth to note that the R^2_{oos} for the specification with two hidden layers also achieves a value of 0.38%, emphasizing the limited marginal benefit in increasing the depth of the network. Surprisingly, feedforward neural networks perform just as well as ENet, which is the most powerful linear specification.

In order to gain more insights into the value of incorporating nonlinear model specifications, this paper includes three powerful deep learning methods: Recurrent Neural Networks (RNN) with LSTM unit layers, RNN with GRU unit layers and Convolutional Neural Networks (CNN). Starting from convolutional neural networks, CNN outperform FNN with an R^2_{oos} of 0.41%, leveraging the additional information captured by the convolutional and pooling layers. However, the model specification that achieved the highest out-of-sample predictive performance is RNN. Specifically, I implement two types of gated RNNs, namely "Long Short-Term Memory" (LSTM) and "Gated Recurrent Unit" (GRU). The former slightly outperforms the latter, with R^2_{oos} values of 0.52% and 0.49%, respectively. These results point to the value of incorporating memory cells that are able to capture long-term dependencies, which are a fundamental characteristic of financial data.

The second and third row of Table 1 distinguish predictability for large stocks (stocks belonging to the top-1000 by market capitalization each month) and small stocks (the bottom-1000 stocks each month). The predictions are taken from the forecasts of the full estimated model (employing all stock observations) and then filtered by market equity each month. The baseline patterns witnessed for the full sample carry over into the two subsamples. Figure 3 provides a visual representation of the comparison in R^2_{oos} statistics between the 3 samples. In particular, OLS keeps performing relatively poorly, regularized linear models represent an improvement, LGBM achives higher predictive performance for the top stock by market capitalization, while the exact opposite holds for XGB. Moreover, among large stocks, deep neural networks exhibit substantial out-performance, with R^2_{oos} ranging from 0.57% for CNN to an astonishing figure of 0.76% for RNN with LSTM cells. Hence, this emphasizes once again the unbeaten predictive performance of deep learning methods, which demonstrate to be extremely successful in forecasting large stocks' excess returns.

Table 2: Annual out-of-sample prediction performance (percentage R^2_{oos})

	OLS +H	OLS-3 +H	PLS	PCR	ENet	RF	LGBM	XGB	FNN (NN1)	LSTM	GRU	CNN
All	27.1	25.6	27.2	24.8	26.7	30.6	36.7	33.8	36.6	32.4	29.3	36.2
Top 1000	24.7	23.1	24.6	22.2	24.1	29.8	34.9	31.7	36.6	30.4	27.5	34.6
Bottom 1000	26.5	24.9	27.0	24.4	26.2	29.1	35.3	32.8	35.0	32.9	29.0	35.8

Note. +H indicates the use of a Huber loss. For Feed Forward Neural Networks, the best model is reported. Estimation ranges from 1 to 5 hidden layers, with NN1 being the most performing model.

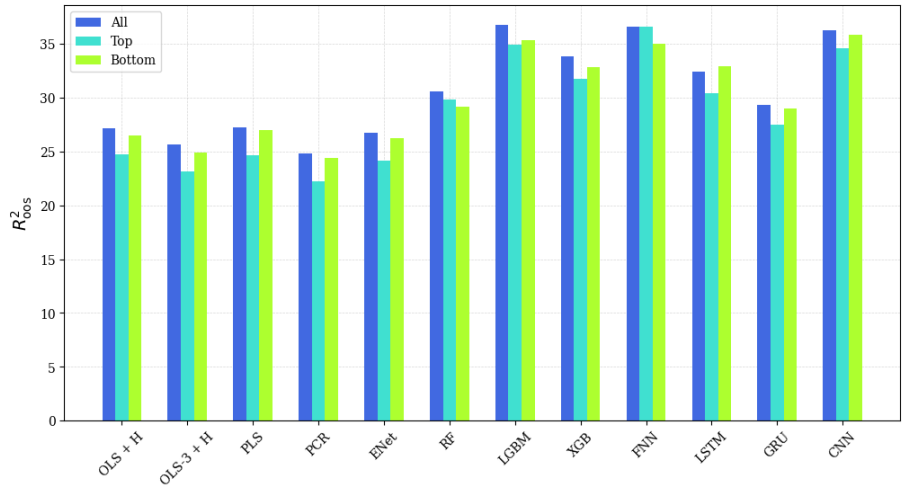

Figure 4: Visual comparison of the R^2_{oos} statistics from Table 2 over the annual horizon

Table 2 and Figure 4 display the results of the analysis conducted at the annual horizon. The comparative performance of the different methods for the annual horizon differs substantially from the monthly. First of all, the annual R^2_{oos} is approximately two order of magnitude larger for all the models. With this set-up non-linear models, such as regression trees and deep neural networks, perform substantially better than linear and regularized-linear models. The biggest difference in performance is indeed encountered for XGB and RF, which produce positive R^2_{oos} figures close to the one of LGBM. The reason behind such a remarkable improvement stands in the ability of machine learning methods to capture patterns that are not merely short-term oriented, but that on the contrary persist over periods of prolonged economic fluctuations.

Table 3: Comparison of monthly out-of-sample prediction using Diebold-Mariano tests

	OLS-3 +H	PLS	PCR	ENet	RF	LGBM	XGB	FNN	LSTM	GRU	CNN
OLS +H	1.06	1.74	1.73	1.85	1.08	1.92	-0.15	1.75	**2.43**	**2.41**	**2.18**
OLS-3 +H		1.83	1.81	1.95	1.06	**2.04**	-0.34	1.82	**2.68**	**2.64**	**2.36**
PLS			-1.03	0.95	-1.16	0.02	**-2.86**	0.26	1.37	1.11	0.92
PCR				**2.12**	-1.05	0.14	**-2.78**	0.56	1.55	1.34	1.15
ENet					-1.25	-0.08	**-2.88**	-0.04	1.20	1.00	0.68
RF						1.41	**-2.47**	1.06	1.56	1.41	1.34
LGBM							**-2.94**	0.06	0.69	0.52	0.37
XGB								**2.64**	**3.18**	**2.88**	**2.97**
FNN									1.13	1.05	0.59
LSTM										-0.55	-1.61
GRU											-0.87

Note. This table reports pairwise Diebold-Mariano test statistics comparing the out-of-sample stock-level prediction performance among thirteen models. Positive numbers indicate the column model outperforms the row model. Bold font indicates the difference is significant at 5% level or better for individual tests. Estimation ranges from 1 to 5 hidden layers, with NN5 being the most performing model.

Table 3 assesses the statistical significance of the difference in model performance at the monthly interval. In particular, Diebold and Mariano (1995) test statistics are reported for a pairwise comparison of a column model versus a row model. The Diebold-Mariano test statistics are standard normally distributed $\mathcal{N}(0,1)$ under the null hypothesis of no difference between the two models. A positive value therefore means that the column model outperforms in forecasting power the row model. Bold numbers denote significance at the 5% level for each individual test.

The first conclusion that we can infer from Table 3 is that OLS is outperformed by almost every model, even though only deep learning methods do so in a statistically significant manner. Moreover, it is interesting to notice that penalized regression (ENet) performs better than dimension reduction techniques, with its improvement over PCR being statistically significant. While XGB is definitely the worst performing model, RF and LGBM do no present particular improvement over linear regularized and dimension reduction models. Deep learning models, such as LSTM, GRU and CNN, are the only models that outperform linear models and XGB with statistical significance. In particular, RNN with LSTM cells outperforms any other model, but with the difference being not statistical significant at the 5% level.

5.2 Variable importance and marginal relationships

I now investigate the relative impact of individual features for the in-sample performance of each model using the measure described in Section 4.7. For each method, I calculate the reduction in R^2 from setting all values of a given predictor to zero within each training sample. The mean across all training samples is computed, creating an overall variable importance measure for each method. These measures are then standardized to sum one, such that comparisons can be made within models. Note that characteristics can exhibit also negative values (equivalent to an improvement in model performance when set equal to zero in the training sample), therefore for some models the sum of the top 20 characteristics could potentially exceed one, to be then

balanced out by the importance variables of the features at the bottom of the ranking. Figure 5 reports the top-20 stock-features for each model according to the variable importance measure.

Figure 5 reports overall rankings of characteristics for all models. For each model I rank the 92 characteristics using the variable importance measure of Figure 5. Their rankings are standardized between 0 and 1, to allow for comparison across models. For each stock feature, the sum of the ranks across all models is computed. The characteristics are then sorted so that the highest total ranks are on top and the lowest ranking characteristics are at the bottom. The color gradient within each column shows the model-specific ranking of characteristics from least to most important (lightest to darkest).

Figure 5 and Figure 6 give an overview of the most influential characteristic for each model. By looking at the individual models of Figure 5, a common pattern cannot be identified easily. While PLS seems to prioritize risk measures such as market beta (beta) and beta squared (betasq), PCR depends heavily on price trends variables such as long-term reversal (mom36m) and short-term reversal (mom1m). ENet, poses particular importance on liquidity variables such as turnover and turnover volatility (turn, std_turn) while also maintaining dependence on short-term reversal (mom1m) as PCR. Moving to deep learning methods, FFN seems to be dominated by fundamental signals such as the growth in long term net operating assets (grltnoa), while for CNN trading variables dominate by importance, namely with the number of zero trading days (zerotrade) and with the dollar tarding volume (dolvol). Finally, RNNs are particularly sensitive to a total return volatility (retvol) in case of LSTM unit layers, and by accounting measures such as capital expenditues an inventory (invest) in case of GRU unit layers. Once again, the advanced ability of LSTM cells to capture temporal dependencies is emphasized its increase in performance when dealing with time-varying variables such as return volatility.

Focusing on Figure 6, the most influential characteristics appear to be size (mvel), asset growth (agr), industry momentum (indmom), long-term reversal (mom36mm), volatility of liquidity (std_dolvol) and dollar volume (dolvol). Hence, a mixture of liquidity varaibles, price variables and fundamental variables identify as most relevant predictors across all models. If linear and regularized linear models appear to identify these variables as more relevant, it is interesting to remark neural networks are driven by a broader range of characteristics, justifying their ability to extrapolate intricate patterns and relationships from the input data set.

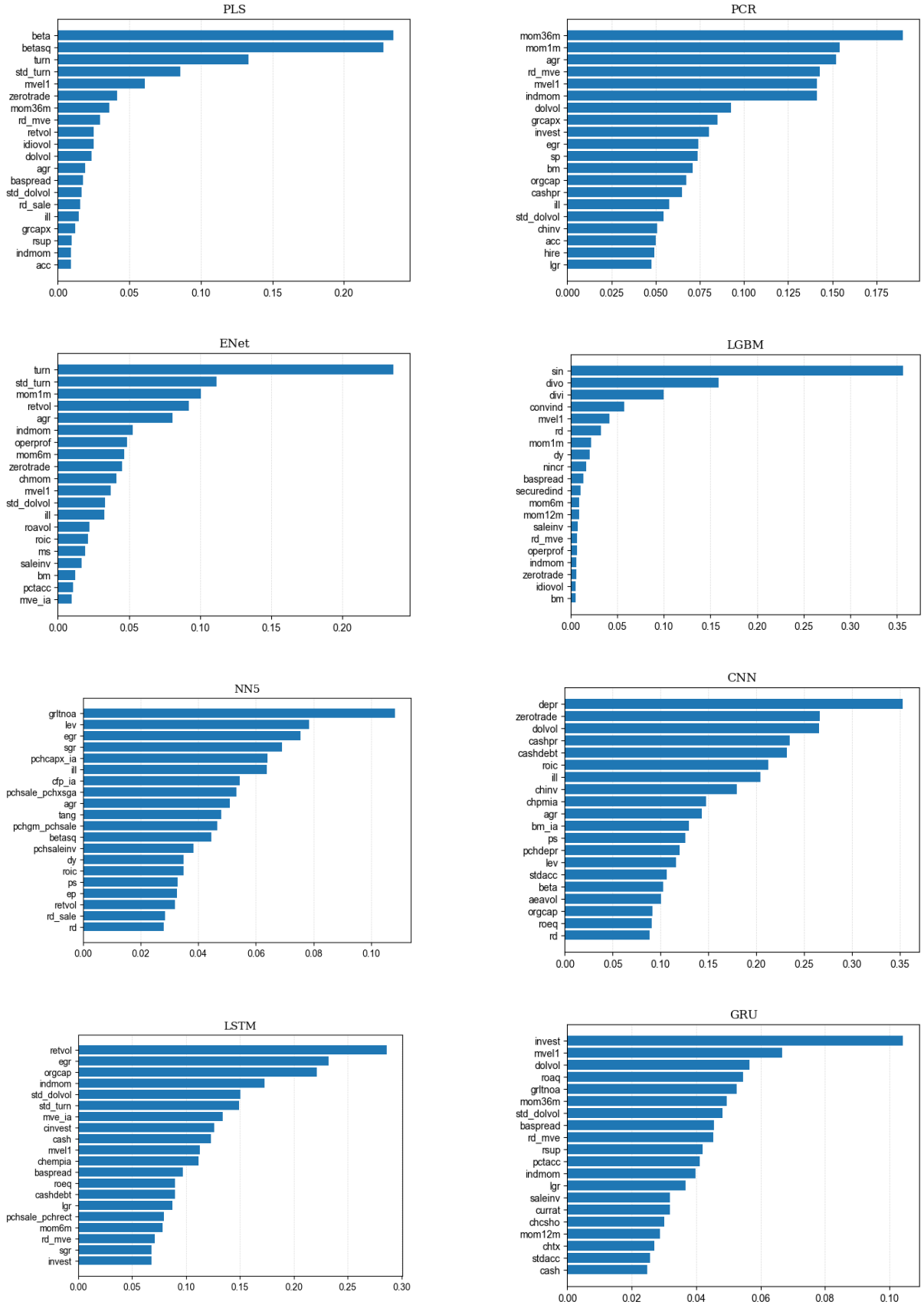

Figure 5: Variable importance for the top-20 most influential variables in each model. Variable importance is an average over all training samples. Variable importance within each model is normalized to sum to one.

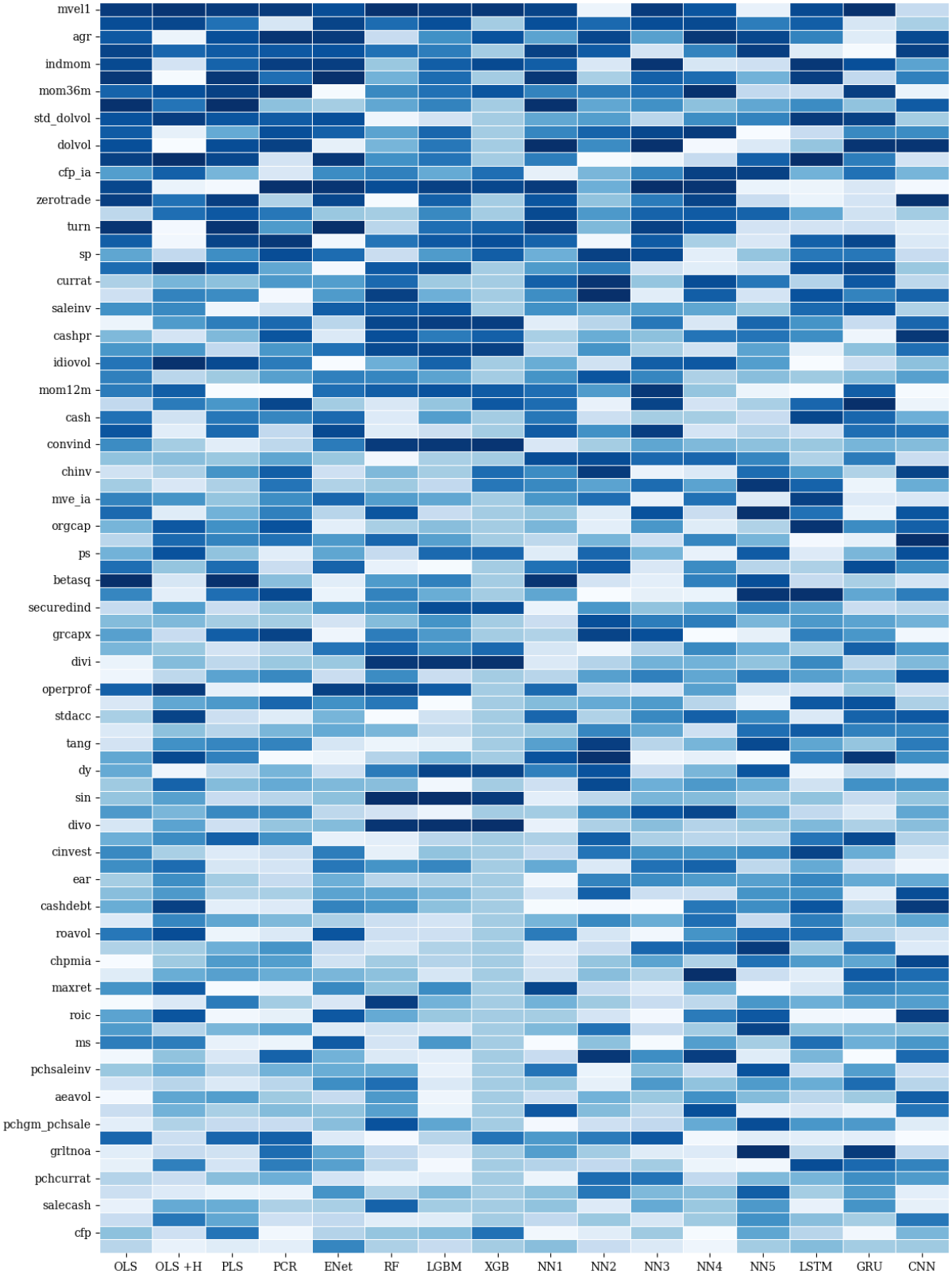

Figure 6: Rankings of ninety-two stock-level characteristics in terms of overall model contribution. Characteristics are ordered based on the sum of their ranks over all models, with the most influential characteristics on the top and the least influential on the bottom. Columns correspond to the individual models, and the color gradients within each column indicate the most influential (dark blue) to the least influential (white) variables.

5.3 Portfolio Forecast Performance

While individual forecast performance has been analyzed extensively in the previous section, the focus is now placed on the comparison of predictive performance of machine learning methods for aggregate portfolio returns. Table 4 and 5 report the result of the analysis using equal-weight portfolios and value-weight portfolios. The results align partly with the analysis carried out previously on individual forecasts. When looking at Table 4, it can be noticed that while for individual forecasts deep learning methods such as LTSM, GRU and CNN perform extremely well, when using the same prediction in order to form a long-short decile strategy, the out-of-sample performance falls drastically. On the other hand, Feedforward neural networks (FNN) perform in line with the expectations, with NN1 being the most powerful model achieving an average realized monthly return of 1.7%. However, the highest risk-adjusted performance is obtained with PLS, which returns on average 1.7% per month (20.4% on an annualized basis), with a monthly volatility of 8% (27.7% annualized) and an annualized out-of-sample Sharpe ratio of 0.74. When comparing this last figure to OLS-3, it can be witnessed an improvement of 72% in terms of Sharpe ratio, definitely outperforming the benchmark. It's also interestinhg to notice that while LGBM outperformed RF and XGB in terms of individual stock returns predictions, now the situation is completely swapped, with XGB and RF achieving out-of-sample Sharpe ratios of 0.59 and 0.45, respectively, against the 0.28 of LGBM.

Table 4: Performance of equal-weight machine learning portfolios

	OLS-3				PLS				PCR			
	Pred	Avg	SD	SR	Pred	Avg	SD	SR	Pred	Avg	SD	SR
Low (L)	0.43	0.89	6.98	0.44	0.02	0.54	6.60	0.28	0.27	0.71	6.74	0.36
High (H)	2.52	1.86	8.13	0.79	2.95	2.25	9.07	0.86	2.65	1.81	8.90	0.71
H - L	2.09	0.96	7.69	0.43	2.92	1.7	8.03	0.74	2.37	1.10	7.98	0.48
	ENet				RF				LGBM			
	Pred	Avg	SD	SR	Pred	Avg	SD	SR	Pred	Avg	SD	SR
Low (L)	0.36	0.55	6.81	0.28	0.81	1.06	7.24	0.51	1.16	1.06	7.01	0.52
High (H)	2.56	1.91	8.46	0.78	1.50	2.14	9.05	0.82	1.45	1.64	7.40	0.77
H - L	2.20	1.36	7.76	0.61	0.69	1.07	8.35	0.45	0.29	0.58	7.32	0.28
	XGB				NN1				NN3			
	Pred	Avg	SD	SR	Pred	Avg	SD	SR	Pred	Avg	SD	SR
Low (L)	1.65	0.79	7.98	0.34	-0.40	0.69	7.81	0.31	0.68	0.70	8.08	0.30
High (H)	3.47	2.21	8.40	0.91	3.38	2.39	8.96	0.92 2.26	1.67	7.64	0.76	
H - L	1.82	1.42	8.31	0.59	3.77	1.7	8.52	0.69	1.57	0.97	7.93	0.42
	GRU				LSTM				CNN			
	Pred	Avg	SD	SR	Pred	Avg	SD	SR	Pred	Avg	SD	SR
Low (L)	0.95	1.38	7.93	0.60	1.19	1.26	6.60	0.66	0.85	1.06	7.05	0.52
High (H)	1.87	1.01	6.77	0.52	1.60	1.37	7.65	0.62	2.14	1.37	7.85	0.60
H - L	0.91	-0.37	7.45	-0.17	0.41	0.11	7.25	0.53	1.29	0.31	7.54	0.14

Note. In this table, we report the performance of prediction-sorted portfolios over the 22-year out-of-sample testing period. All stocks are sorted into deciles based on their predicted returns for the next month. Columns "Pred," "Avg," "SD," and "SR" provide the predicted monthly returns for each decile, the average realized monthly returns, their standard deviations, and Sharpe ratios, respectively. All portfolios are equal weighted.

Table 5 reports the analysis performed on value-weight machine learning portfolios. The first thing that is evident is the fall in performance when using value-weights. This is likely to be

caused by the fact that the statistical objective functions employed in the prediction procedure minimize equally weighted forecast errors. In this scenario, NN1 is still the best performing model with an out-of-sample Sharpe ratio of 0.45. RNN and CNN still perform quite badly, being outperformed by linear models such as OLS and ENet.

Table 5: Performance of value-weigth machine learning portfolios

	OLS-3				PLS				PCR			
	Pred	Avg	SD	SR	Pred	Avg	SD	SR	Pred	Avg	SD	SR
Low (L)	0.41	0.66	5.49	0.41	0.02	0.62	5.83	0.37	0.27	0.61	5.25	0.40
High (H)	2.45	1.41	8.58	0.57	2.81	0.97	8.26	0.41	2.55	0.96	8.44	0.40
H - L	2.03	0.75	7.26	0.36	2.80	0.35	7.17	0.17	2.27	0.35	7.05	0.18
	ENet				RF				LGBM			
	Pred	Avg	SD	SR	Pred	Avg	SD	SR	Pred	Avg	SD	SR
Low (L)	0.35	0.54	5.59	0.33	0.82	0.57	5.98	0.33	1.16	0.78	5.06	0.53
High (H)	2.47	1.29	7.46	0.60	1.45	1.15	8.06	0.50	1.39	0.96	5.73	0.58
H - L	2.12	0.74	6.64	0.39	0.63	0.58	7.13	0.28	0.22	0.18	5.46	0.11
	XGB				NN1				NN3			
	Pred	Avg	SD	SR	Pred	Avg	SD	SR	Pred	Avg	SD	SR
Low (L)	1.66	0.66	6.51	0.35	-0.38	0.30	7.76	0.13	0.75	0.32	7.69	0.14
High (H)	3.15	1.14	6.95	0.57	3.16	1.51	10.50	0.50	2.16	0.83	6.55	0.44
H - L	1.49	0.48	6.78	0.25	3.54	1.21	9.25	0.45	1.41	0.51	7.15	0.25
	GRU				LSTM				CNN			
	Pred	Avg	SD	SR	Pred	Avg	SD	SR	Pred	Avg	SD	SR
Low (L)	0.98	0.72	5.40	0.46	1.19	0.44	5.70	0.27	0.87	0.58	5.93	0.34
High (H)	1.87	0.98	5.56	0.61	1.59	0.71	5.88	0.42	2.20	0.93	6.66	0.48
H - L	0.89	0.26	5.53	0.16	0.40	0.27	5.81	0.16	1.33	0.35	6.33	0.19

Note. In this table, we report the performance of prediction-sorted portfolios over the 22-year out-of-sample testing period. All stocks are sorted into deciles based on their predicted returns for the next month. Columns "Pred," "Avg," "SD," and "SR" provide the predicted monthly returns for each decile, the average realized monthly returns, their standard deviations, and Sharpe ratios, respectively. All portfolios are value weighted.

Finally, it is interesting to see how taking a short position in the bottom decile, for every method in both the portfolio configurations, always leads to sacrificing part of the positive returns achieved by taking a long position in the top decile. This may be caused by the fact that in general terms, if we exclude the Dot-com bubble of the early 2000s and the Recessions of 2008-2009 and 2020, the stock market has has always performed positively in the twentieth century. The out-of-sample period taken into consideration definitely overlaps with this positive period ofr the market, possibly pointing at the cause of underperformance when shorting the bottom decile.

6 Conclusion

The main question addressed in this thesis is whether recurrent neural retworks (RNN) with LSTM and GRU cells and convolutional neural networks (CNN) improve the predictive performance of excess stock returns compared to traditional asset pricing models.

By performing a comparative analysis of several machine learning methods, it can be concluded that the aforementioned deep learning methods are among the best performing model

in predicting risk premia. The economic justification of this phenomenon standing in their innate ability of capturing non-linear interactions among predictors and long-term dependencies in the data. However, even though they outperform all the other machine learning methods analysed, this improvement is only statistically significant with respect to OLS. Furthermore, when building long-short decile spread portfolios using simple return predictions, feedforward neural networks provide the best Sharpe ratio performance, while more advanced deep learning methods such as RNN and CNN are among the worst performing models. This means that good forecasting performance on individual excess returns does not necessarily imply good portfolio performance, given the two different problem structures. Moreover, I also find that while linear models agree on a small set of dominant predictive signals, such as liquidity variables, price trends and fundamental ratios, deep learning methods draw form a wider set of characteristic variables. This reinforces the idea that, unlike the other machine learning methods, deep learning networks are able to construct predictions by extrapolating intricate features' patterns from several sources inside the data.

In summary, machine learning techniques represent an added value to the field of asset pricing in the context of risk premia prediction. In order to further establish their predictive power, these techniques should be tested for other markets, time frames or by using different model architectures. Further research could be dedicated to their application in the construction of portfolios that take into account of real market frictions, such as for instance trading costs. Lastly, given their high adaptability in predicting risk premia, these techniques could be further tested to forecast the assets' market volatility.

References

Breiman, L., Friedman, J., Stone, C., & Olshen, R. (1984). *Classification and regression trees*. Taylor & Francis.

Breiman, L. (1996). Bagging predictors. *Machine learning*, *24*(2), 123–140.

Breiman, L. (2001). Random forests. *Machine learning*, *45*(1), 5–32.

Bryzgalova, S., Pelger, M., & Zhu, J. (2021). Forest through the trees: Building cross-sections of stock returns, available at SSRN: `https://ssrn.com/abstract=3493458`.

Bryzgalova, S., Lerner, S., Lettau, M., & Pelger, M. (2022). Missing financial data. *SSRN*. `https://ssrn.com/abstract=4106794`

Campbell, J. Y., & Shiller, R. J. (1988a). The dividend-price ratio and expectations of future dividends and discount factors. *The Review of Financial Studies*, *1*(3), 195–228.

Campbell, J. Y., & Shiller, R. J. (1988b). Stock prices, earnings, and expected dividends. *The Journal of Finance*, *43*(3), 661–676.

Campbell, J. Y., & Thompson, S. B. (2008). Predicting excess stock returns out of sample: Can anything beat the historical average? *The Review of Financial Studies*, *21*(4), 1509–1531.

Chen, L., Pelger, M., & Zhu, J. (2023). Deep learning in asset pricing. *Management Science*, available at SSRN: `https://ssrn.com/abstract=3350138`..

Cho, K., van Merrienboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning phrase representations using rnn encoder-decoder for statistical machine translation.

Chung, J., Gulcehre, C., Cho, K., & Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. *NIPS'2014 Deep Learning Workshop*.

Chung, J., Gülçehre, Ç., Cho, K., & Bengio, Y. (2015). Gated feedback recurrent neural networks. *International Conference on Machine Learning (ICML)*.

Cochrane, J. H. (2008). The dog that did not bark: A defense of return predictability. *Review of Financial Studies*, *21*(4), 1533–1575.

Cong, L., Tang, K., Wang, J., & Zhang, Y. (2021). Alphaportfolio: Direct construction through deep reinforcement learning and interpretable ai. `https://ssrn.com/abstract=3554486`

Diebold, F. X., & Mariano, R. S. (1995). Comparing predictive accuracy. *Journal of Business & Economic Statistics*, *13*, 134–144.

Fama, E. (1970). Efficient capital markets: A review of theory and empirical work. *The Journal of Finance*, *25*(2), 383–417.

Fama, E., & French, K. (1988). Dividend yields and expected stock returns. *Journal of Financial Economics*, *22*(1), 3–25.

Fama, E., & French, K. (1993). Common risk factors in the returns on stocks and bonds. *Journal of Financial Economics*, *33*, 3–56.

Fama, E., & French, K. (2015). A five-factor asset pricing model. *Journal of Financial Economics*, *116*(1), 1–22.

Feng, G., He, J., & Polson, N. G. (2018). Deep learning for predicting asset returns.

Freund, Y., Schapire, R., & Abe, N. (1999). A short introduction to boosting. *Journal- Japanese Society For Artificial Intelligence*, *14*(771-780), 1612.

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

Gu, S., Kelly, B., & Xiu, D. (2020). Empiricial asset pricing via machine learning. *The Review of Financial Studies*, *33*(5), 2223–2273.

Gu, S., Kelly, B., & Xiu, D. (2021). Autoencoder asset pricing models. *Journal of Econometrics*, *222*(1), 429–450.

Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, *9*(8), 1735–1780.

Hoerl, A. E., & Kennard, R. W. (1970). Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, *12*(1), 55–67.

Hornik, K., Stinchcombe, M., & White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural networks*, *2*(5), 359–366.

Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift.

Jegadeesh, N., & Titman, S. (1993). Returns to buying winners and selling losers: Implications for stock market efficiency. *The Journal of Finance*, *48*(1), 65–91.

Jozefowicz, R., Zaremba, W., & Sutskever, I. (2015). An empirical evaluation of recurrent network architectures. *International Conference on Machine Learning (ICML)*.

Kelly, B., Pruitt, S., & Su, Y. (2019). Characteristics are covariances: A unified model of risk and return. *Journal of Financial Economics*.

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.

Kelly, B. T., Pruitt, S., & Su, Y. (2020). Instrumented principal component analysis. https://ssrn.com/abstract=2983919

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

LeCun, Y. (1989). *Generalization and network design strategies* (tech. rep. CRG-TR-89-4). University of Toronto.

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), 13–37.

Markowitz, H. (1952). Portfolio selection. *Journal of Finance*, *7*(1), 77–91.

Rapach, D. E., et al. (2010). Out-of-sample equity premium prediction: Combination forecasts and links to the real economy. *The Review of Financial Studies*, *23*(2), 821–862.

Rosenberg, B., Reid, K., & Lanstein, R. (1985). Persuasive evidence of market inefficiency. *Journal of Portfolio Management*, *11*, 9–17.

Sharpe, W. F. (1964). Capital asset prices: A theory of market equilibrium under conditions of risk. *Journal of Finance*, *19*(3), 425–442.

Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, *58*(1), 267–288.

Welch, I., & Goyal, A. (2008). *Review of Financial Studies*, *21*(4), 1455–1508.

Appendix

Table 6: Monthly out-of-sample prediction performance using raw returns as target variable (percentage R^2_{oos})

	OLS +H	OLS-3 +H	PLS	PCR	ENet	RF	LGBM	XGB	FNN	LSTM	GRU	CNN
All	-0.20	-0.14	0.44	0.41	0.47	-0.33	0.39	-0.67	0.45	0.53	0.44	0.48
Top 1000	0.10	0.18	0.40	0.44	0.47	-0.96	0.51	-1.36	0.66	0.70	0.52	0.60
Bottom 1000	-0.31	-0.23	0.48	0.43	0.49	-0.14	0.40	-0.23	0.42	0.50	0.46	0.47

Note. +H indicates the use of a Huber loss. For Feed Forward Neural Networks, the best model is reported, with the estimation ranging from 1 to 5 hidden layers.

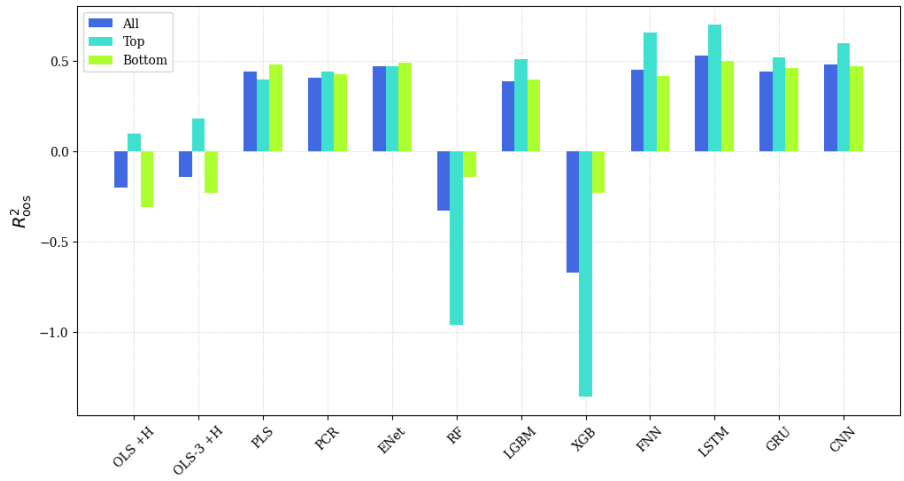

Figure 7: Visual comparison of the R^2_{oos} statistics from Table 3 using raw returns as target variable

Programming code

The programming code has been uploaded in the file "Final Main.zip". The code consists of the following folders:

- Data Preparation: this folder contains a Python noteboook file in which the data is pre-processed in order to be fed properly to the Main code.
- Code: this folder contains all the Python file in which the machine learning models are performed and predictions are computed. The main.py file represents the core of the analysis, with all the models specifications. Other files include some classes for the different typed of models, such as NN.py for neural networks for instance.
- Analysis Results: this folder contains two Pyhton notebooks in which further analyses, such as the portfolio formation or the Diebold Mariano tests, are performed.