AUGMENTING TRANSFERRED REPRESENTATIONS FOR STOCK CLASSIFICATION

Elizabeth Fons[1] *Paula Dawson*[2] *Xiao-jun Zeng*[1] *John Keane*[1] *Alexandros Iosifidis*[3]

[1] Department of Computer Science, University of Manchester, UK.
[2]AllianceBernstein, London, UK.
[3]Department of Electrical and Computer Engineering, Aarhus University, Denmark.

ABSTRACT

Stock classification is a challenging task due to high levels of noise and volatility of stocks returns. In this paper we show that using transfer learning can help with this task, by pre-training a model to extract universal features on the full universe of stocks of the S&P500 index and then transferring it to another model to directly learn a trading rule. Transferred models present more than double the risk-adjusted returns than their counterparts trained from zero. In addition, we propose the use of data augmentation on the feature space defined as the output of a pre-trained model (*i.e*. augmenting the aggregated time-series representation). We compare this augmentation approach with the standard one, i.e. augmenting the time-series in the input space. We show that augmentation methods on the feature space leads to 20% increase in risk-adjusted return compared to a model trained with transfer learning but without augmentation.

***Index Terms*—** Transfer learning, data augmentation, deep learning, financial signal processing, stock classification

1. INTRODUCTION

Stock market prediction is a challenging task primarily driven by a high degree of noise and volatility influenced by external factors such as extreme macroeconomic conditions, heightened correlations across multiple markets, and investor's behaviour. While much work has been done on stock movement prediction [1–3], it remains an open research challenge. For the past decade, deep neural networks have exhibited very good performance in many different fields such as computer vision, natural language processing and robotics. In recent years, it has been shown that deep learning is capable of identifying nonlinearities in time series data leading to profitable strategies [4–6].

A common approach when using deep learning for stock classification is to focus on developing a model that predicts the movement of an index or of stocks, and then build a simple trading rule in order to test the method's profitability (in some cases not taking into account transaction costs that could potentially erode some or all earnings) [4, 7–9]. Further, little work has been done using large-scale datasets such as all S&P500 constituents in a survivorship bias-free way - survivorship bias is the tendency to view performance of existing stocks as a representative sample without regarding those that have gone bankrupt and leads to overestimation of historical performance [10]. In general, previous work has focused on predicting either movement of an index or of a small number of stocks [5].

In this work, we present a model that can learn a trading rule directly from a large-scale stock dataset. For this, we propose using transfer learning, where we pre-train a model with past returns of all constituent stocks of the S&P500 index, and then transfer it and fine-tune it on a dataset that has the trading rule included. Transfer learning is widely used in computer vision when the target dataset contains insufficient labeled data [11, 12]; to our knowledge, it has been rarely used in deep learning models for time series data [13]. Current approaches in trading strategies treat each market or asset in isolation, with few use cases of transfer learning in the financial literature [1, 14].

Motivated by increasing interest in data augmentation for time series, we propose using data augmentation to improve generalisation [15, 16]. Given that we are using transfer learning, we propose transforming the vector representation of data within the learned feature space (*i.e*. augmenting the aggregated time series representation obtained from the output of the pre-trained model, following DeVries *et al*. [17]) and compare this with standard input space augmentation, done by applying transformations such as time warp, jittering, etc. [18, 19].

The contributions of the paper are as follows:

- We pre-train a model using all the constituents of the S&P500 index to extract universal features and then we transfer this model to another to learn a trading rule.
- We propose the use of data augmentation on the feature space defined as the output of the pre-trained model and we compare this approach with the standard augmentation in the input space.
- We test our model by building the learned trading rule and calculate profitability taking into account transaction fees.

2. METHODOLOGY

2.1. Dataset

The data used in this study consists of the daily returns of all constituent stocks of the S&P500 index, from 1990 to 2018. It comprises 7000 trading days, and approximately 500 stocks per day. We use the data pre-processing scheme proposed by Krauss *et al*. [4], where the data is divided into splits of 1000 days, with a sliding window of 250 days. Each split overlaps with the previous one by 750 points, resulting in 25 splits in total; a model is trained on each split. Inside each of the 25 splits, the data is segmented into sequences consisting on 240 time steps $\{\tilde{R}^s_{t-239}, \ldots, \tilde{R}^s_t\}$ for each stock s, with a sliding window of one day, as shown in Figure 1. The first 750 days consist of the training set, and the test set is formed by

This work was supported by the European Union's Horizon 2020 research and innovation programme under the Marie Sklodowska-Curie Grant Agreement no. 675044 (http://bigdatafinance.eu/), Training for Big Data in Financial Research and Risk Management. A. Iosifidis acknowledges funding from the Independent Research Fund Denmark project DISPA (Project Number: 9041-00004).

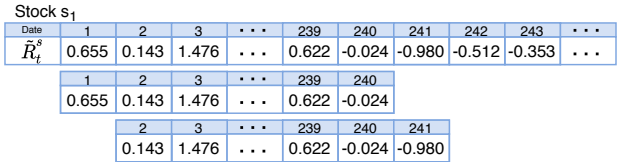

Fig. 1: Construction of input sequences, segmented in 240 time steps, with a moving window of one day.

the last 250 days. This leads to a training with approximately 255K samples ((750-240)*500) and the test set with approximately 125K samples. The data is standardised by subtracting the mean of the training set (μ_{train}) and dividing by the standard deviation (σ_{train}): $\tilde{R}_t^s = \frac{R_t^s - \mu_{train}}{\sigma_{train}}$, with R_t^s being the return of stock s at time t.

2.2. Trading rule and training targets

A goal of this work is to evaluate whether training a neural network by directly passing the trading rule as the classification target is better than training a binary classifier and then applying a trading rule. Therefore, to construct the target, taking the ≈ 500 daily stocks, we ranked them by their returns and the top K are labeled as *buy*, the bottom K are labeled as *sell*, and the rest as *do nothing*. Following the recommendation from Krauss *et al*. [4], we use $K = 10$. This leads to a highly imbalanced dataset, with a label proportion of $10 : 10 : 480$.

The source network is trained as a binary classification task, where the target variable Y_{t+1}^s for stock s and date t can take two values, 1 if the returns are above the daily median (trend up) and 0 if returns are below the daily median (trend down).

2.3. Architecture and training

The network architecture selected for transfer learning (source network) is a one layer LSTM proposed by Krauss *et al*. [4]. We chose this architecture because it achieved good performance on a large, liquid dataset - with similarities to the S&P500. The network is a single layer LSTM with 25 neurons, and a fully connected two-neuron output. We use a learning rate of 0.001, batch size 128 and early stopping with patience 10 with RMSProp as optimizer. We implemented the neural network architectures using pytorch [20] in Python.

After training the source network on the 25 splits of data, we have 25 neural networks. We then remove the output layer and replace it with a fully connected layer of n neurons and an output layer of 3 neurons. The weights of the LSTM layer are fixed (there is no retraining), and the fully connected layer and output are trained with the new target data that incorporates the trading rule. Fawaz *et al*. [13] proposes fine-tuning the transferred model parameters on the new dataset, but this leads to poor performance in our case, so the weights on the LSTM are left fixed, and just the new added layers are trained.

The loss used for training in all cases is the cross-entropy loss. An alternative approach is to incorporate a loss term that takes into account the direct information on the positions from the network, and optimizes the average return, as follows:

$$\mathcal{L}_{R+CE}(\mathbf{\Theta}) = \mathcal{L}_{CE} + \alpha \mathcal{L}_{returns} = \mathcal{L}_{CE} + -\alpha \frac{1}{B} \sum R(i,t) \quad (1)$$

where B is the size of the batch, $R(i,t)$ is the return captured by the network prediction for asset i at time t and α is a scaling factor so both terms are equally represented.

As the dataset is highly imbalanced, we sample the training data with higher probability in the minority classes, obtaining a balanced representation of the classes in each batch.

2.4. Augmentation

The approximately 255K samples of the training set are divided into training and validation with a proportion $80/20$. The validation set is used for early stopping the training. Each train set is augmented one time, *i.e*. we apply an augmentation method to the training data and the final set corresponds to the original data plus the augmented one, doubling the amount of samples. We study two forms of data augmentation: applying random transformations on the input data and doing data augmentation on the feature vector obtained by evaluating the input on the fixed LSTM layer. Figure 2 shows the training process with transfer learning and both forms of data augmentation.

2.4.1. Data augmentation in feature space

Data augmentation in the feature space as proposed by DeVries *et al*. [17] allows new samples to be generated in a domain-agnostic approach. It is an intuitive and direct approach where each sample is projected into feature space by feeding it through the fixed LSTM layer and then applying a transformation to the transformed vector (sometimes called context vector). The methods used to augment data in the feature space are described as follows and an example is shown in Figure 3a:

Interpolation: for each sample in the dataset, we find its K intra-class nearest neighbours in feature space. For each pair of neighbouring vectors, a new vector is generated using interpolation:

$$\mathbf{c}_j' = (\mathbf{c}_k - \mathbf{c}_j)\lambda + \mathbf{c}_j$$

where $\mathbf{c}_j'$ is the new vector, $\mathbf{c}_k$ and $\mathbf{c}_j$ are neighboring vectors and λ is a variable in the range $\{0, 1\}$. We used $\lambda = 0.2$.

Extrapolation: similarly, we apply extrapolation to the feature space vectors in the following way:

$$\mathbf{c}_j' = (\mathbf{c}_j - \mathbf{c}_k)\lambda + \mathbf{c}_j$$

In this case, λ is in the range $\{0, \infty\}$. We used $\lambda = 0.2$.

Random noise (noise): Gaussian noise is generated with zero mean and per-element standard deviation calculated across all transformed vectors in the dataset; the noise is scaled by a global parameter γ:

$$\mathbf{c}_i' = \mathbf{c}_i + \gamma X, X \sim \mathcal{N}\{0, \sigma_i^2\}$$

Jittering (Jit-feat): Random noise with mean $\mu = 0$ and standard deviation $\sigma = 0.05$ is added to the context vector.

2.4.2. Data augmentation in input space

Most cases of time series data augmentation correspond to random transformations in the magnitude and time domain, such as jittering (adding noise), slicing, permutation (rearranging slices) and magnitude warping (smooth element-wise magnitude change). The following methods were used for evaluation and are shown in Figure 3b.

Magnify: a variation of window slicing proposed by Le Guennec et al [21]. In window slicing, a window of 90% of the original time series is selected at random. Instead, here we randomly slice windows between 40% and 80% of the original time series, but always from the fixed end of the time series (*i.e*. we slice the beginning of

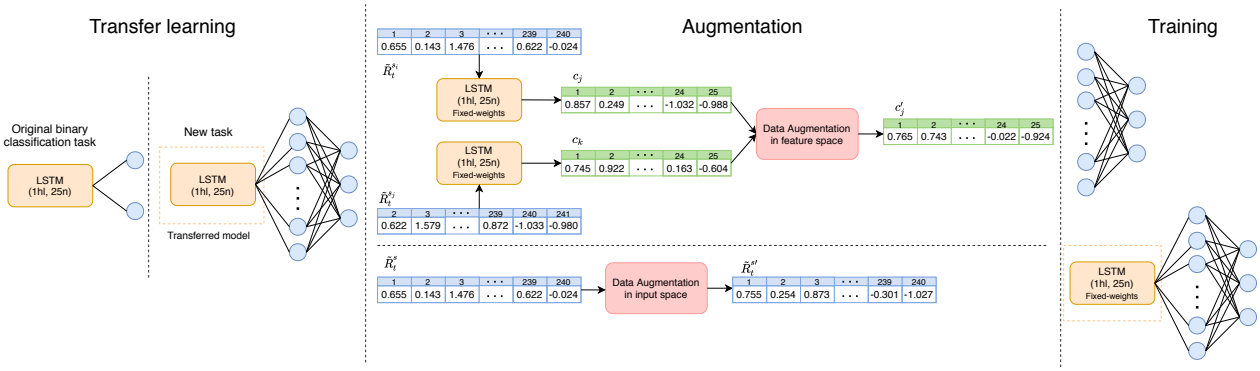

Fig. 2: General training process with transfer learning and data augmentation. The panel on the left shows the pre-trained model and the new transferred one. On the centre, workflow of the two augmentation approaches, data augmentation on the feature space (top) and data augmentation on the input space (bottom); on the right, the resulting networks to be trained.

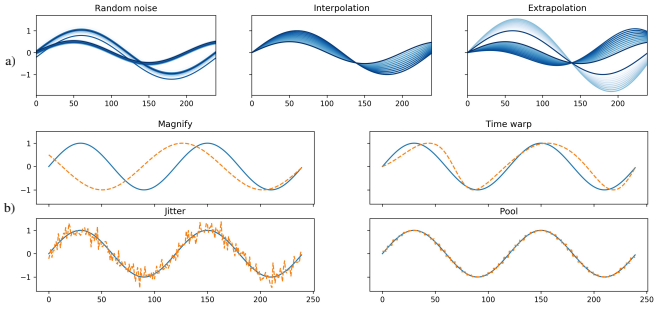

Fig. 3: a) Examples of feature space augmentation methods on a sine wave. Random noise added with $\gamma = 0.3$, interpolation and extrapolation between two sinusoids for values of λ between 0 and 1. b) Examples of input augmentation methods on a sine wave. Blue line corresponds to the original series and the dotted orange lines correspond to the augmented pattern.

the time series by a random factor). Randomly selecting the starting point of the slicing would make sense in an anomaly detection framework, but not on a trend prediction problem as is our case. The resulting time series is interpolated to the original size.

Jittering (Jit-inp): Gaussian noise with a mean $\mu = 0$ and standard deviation $\sigma = 0.05$ is added to the time series [19]. This is analogous to the random noise method applied to the context vectors.

Pool: Reduces the temporal resolution without changing the length of the time series by averaging a pooling window. We use a window of size 3. This method is inspired by the resizing data augmentation process followed in computer vision.

3. EVALUATION

3.1. Transfer learning

We test transfer learning on two networks, one with a fully connected layer of 25 neurons and one with 100 neurons, and compare the proposed networks trained using transfer learning with the same topology trained from scratch. In order to avoid random initialization conflicts on the non-transferred networks, we train three separate instances with different initial weight values (*i.e.* we use three different seeds on the random initialisation) and average their performance.

Given that we want to evaluate financial performance, we build a portfolio by ranking the output of the networks labeled class 1 (*buy*) and 2 (*sell*); we then take the 10 with highest probability for each class and build a long-short portfolio. Portfolios are analysed after transaction costs of 5bps per trade. The portfolio performance metric we use is Information ratio (IR) - the ratio between excess return (portfolio returns minus benchmark returns) and tracking error (standard deviation of excess returns) [22]. As the portfolios are long-short, they are market-neutral, therefore, performance of the portfolio in independent of performance of the market and no benchmark has to be subtracted. We also calculate the downside information ratio - the ratio between excess return and the downside risk (variability of under-performance below the benchmark), that differentiates harmful volatility from total overall volatility. We compare our results with Krauss *et al.* [4], in which a binary classifier is trained and then the trading rule is applied. We also calculate two classification metrics, accuracy and macro-F1 expressed by the mean and standard deviation over the 25 data splits.

Table 1 shows the performance of the models trained using only the cross-entropy loss with and without transfer learning, as well as training with the combined loss of cross-entropy and return maximization from equation 1. We see that, in all cases, the models trained from scratch show a poor financial performance as shown by the information ratio. All transferred learned models show an equal or higher performance than the baseline LSTM method, with the models trained with the combined loss having a slightly higher performance. The classification metrics are higher for models without transfer learning and trained only on the cross-entropy loss. Using the combined loss hurts classification metrics in all cases, but as expected, it improves portfolio performance.

3.2. Data augmentation

Table 2 shows the performance of augmentation methods for the architectures with a fully connected layer of 25 neurons and table 3 models with a fully connected layer of 100 neurons. All models were trained using the combined loss of equation 1, unless stated otherwise. In both topologies, augmentation on the input space is ineffective, and in most cases it decreases performance with respect to the model trained with transfer learning but without augmentation. In contrast, models trained with augmentation on the feature space tend to improve on IR, in the case of jittering.

Table 1: Performance of the $k = 10$ long-short portfolios after transaction costs, built from models trained from zero and models whose weights where transferred from a pre-trained model.

Method	Ann ret	Ann vol	IR	D. Risk	DIR	Acc	Macro-F1
LSTM [4]	29.2	28.66	1.02	19.08	1.53	—	—
No TL (25)+$\mathcal{L}_{CE}$	12.99	38.15	0.34	25.48	0.52	73.13±18.94	$\mathbf{33.57 \pm 6.5}$
No TL (25)+$\mathcal{L}_{R+CE}$	19.58	38.9	0.51	25.9	0.77	59.64±20.48	29.21±6.98
TL+FC(25)+$\mathcal{L}_{CE}$	32.25	30.29	1.06	19.6	1.65	68.34±16.5	31.79±5.12
TL+FC(25)+$\mathcal{L}_{R+CE}$	**34.62**	30.2	**1.15**	19.59	**1.77**	64.79±16.86	30.72±5.28
No TL (100)+$\mathcal{L}_{CE}$	5.05	42.60	0.12	29.28	0.19	$\mathbf{74.69 \pm 21.14}$	33.37±6.94
No TL (100)+$\mathcal{L}_{R+CE}$	21.05	39.95	0.55	25.9	0.84	57.02 ± 21.95	28.24 ± 8.12
TL+FC(100)+$\mathcal{L}_{CE}$	30.83	30.31	1.02	19.79	1.56	68.88±15.93	31.95±4.76
TL+FC(100)+$\mathcal{L}_{R+CE}$	32.14	29.97	1.07	19.87	1.62	64.72±17.25	30.7±5.41

Table 2: Performance of the $k = 10$ long-short portfolios after transaction costs, for the TL+FC(25) model trained with different augmentation methods and the combined loss $\mathcal{L}_{R+CE}$.

Method	Ann ret	Ann vol	IR	D. Risk	DIR	Acc	Macro-F1
LSTM [4]	29.2	**28.66**	1.02	19.08	1.53	—	—
No TL (25)+$\mathcal{L}_{CE}$	12.99	38.15	0.34	25.48	0.52	73.13±18.94	$\mathbf{33.57 \pm 6.5}$
TL+FC(25)+$\mathcal{L}_{CE}$	32.25	30.29	1.06	19.6	1.65	68.34±16.5	31.79±5.12
TL+FC(25)+$\mathcal{L}_{R+CE}$	34.62	30.20	1.15	19.59	1.77	64.79±16.86	30.72±5.28
TL+FC(25) Extrapolation	**39.70**	29.43	**1.35**	18.96	**2.09**	62.90±17.87	30.10±5.81
TL+FC(25) Interpolation	36.87	29.69	1.24	**18.93**	1.95	62.46±17.80	29.95±5.74
TL+FC(25) Noise	30.97	29.15	1.06	19.14	1.62	62.43±18.12	29.95±5.81
TL+FC(25) Jitter-feat	39.11	29.93	1.31	19.22	2.03	62.71±17.84	30.04±5.71
TL+FC(25) Jitter-input	29.74	39.94	0.96	20.12	1.48	68.23±16.62	31.75±5.06
TL+FC(25) Magnify	20.39	29.41	0.69	19.86	1.03	63.78±16.78	30.42±5.47
TL+FC(25) Pool	27.18	29.96	0.91	19.64	1.38	57.71±17.43	28.38±5.72
TL+FC(25) Time Warp	32.76	29.46	1.11	19.21	1.71	61.81±19.96	29.80±5.48

Table 3: Performance of the $k = 10$ long-short portfolios after transaction costs, for the TL+FC(100) model trained with different augmentation methods and the combined loss $\mathcal{L}_{R+CE}$.

Method	Ann ret	Ann vol	IR	D. Risk	DIR	Acc	Macro-F1
LSTM [4]	29.2	28.66	1.02	19.08	1.53	—	—
No TL (100)+$\mathcal{L}_{R+CE}$	21.05	39.95	0.55	25.9	0.84	57.02±21.95	28.24±8.12
TL+FC(100)+$\mathcal{L}_{CE}$	30.83	30.31	1.02	19.79	1.56	68.88±15.93	31.95±4.76
TL+FC(100)+$\mathcal{L}_{R+CE}$	32.14	29.97	1.07	19.87	1.62	64.72±17.25	30.7±5.41
TL+FC(100) Extrapolation	27.38	29.33	0.93	19.42	1.41	62.74±17.73	30.05±5.81
TL+FC(100) Interpolation	30.84	29.72	1.04	19.38	1.59	62.49±17.35	30.01±5.63
TL+FC(100) Noise	29.02	29.44	0.99	19.2	1.51	62.2±17.82	29.87±5.73
TL+FC(100) Jitter-feat	**37.14**	29.31	**1.27**	**18.92**	**1.96**	61.84±17.89	29.75±5.75
TL+FC(100) Jitter-input	29.49	30.32	0.97	19.73	1.49	67.79±17.18	31.64±5.46
TL+FC(100) Magnify	22.11	30.36	0.73	20.21	1.09	67.12±16.68	30.34±5.27
TL+FC(100) Pool	27.64	29.50	0.94	18.98	1.46	57.64±17.89	28.37±5.89
TL+FC(100) Time Warp	26.64	29.65	0.90	19.49	1.37	65.55±18.03	29.69±5.89

4. CONCLUSIONS

In this work, we have shown that using transfer learning on a stock classification task where a trading rule is included in the training dataset improves financial performance when compared to training a neural network from scratch. To evidence this we built long-short portfolios following the proposed trading rule. Models trained with transfer learning improve information ratio by more than double with respect to models trained without a source model. We also showed that using a training loss that combines a classification objective with maximization of returns improves risk adjusted returns when compared with the single cross-entropy loss.

Finally, we investigated the use of data augmentation on the feature space (defined as the output of the pre-trained model) and compared it with traditional data augmentation methods on the input space. Augmentation on the input space improves up to 20% risk adjusted returns when compared to a transferred model without augmentation. While in this paper we studied data augmentation on the feature space, it would be of interest to further investigate the learned representations of the pre-trained model. A future research topic will be to investigate the explainability of the learned representations on the feature space.

5. REFERENCES

[1] Zihao Zhang, Stefan Zohren, and Stephen Roberts, "Deeplob: Deep convolutional neural networks for limit order books," *IEEE Transactions on Signal Processing*, vol. 67, 08 2018.

[2] D. T. Tran, A. Iosifidis, J. Kanniainen, and M. Gabbouj, "Temporal attention-augmented bilinear network for financial time-series data analysis," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 30, no. 5, pp. 1407–1418, 2019.

[3] Nikolaos Passalis, Anastasios Tefas, Juho Kanniainen, Moncef Gabbouj, and Alexandros Iosifidis, "Temporal bag-of-features learning for predicting mid price movements using high frequency limit order book data," *IEEE Transactions on Emerging Topics in Computational Intelligence*, 10 2018.

[4] Christopher Krauss, Xuan Anh Do, and Nicolas Huck, "Deep neural networks, gradient-boosted trees, random forests: Statistical arbitrage on the $s\&p$ 500," *European Journal of Operational Research*, vol. 259, no. 2, pp. 689 – 702, 2017.

[5] Thomas Fischer and Christopher Krauss, "Deep learning with long short-term memory networks for financial market predictions," *European Journal of Operational Research*, vol. 270, no. 2, pp. 654–669, 2018.

[6] Mininder Sethi, Philip Treleaven, and Sebastian Rollin, "Beating the $s\&p$ 500 index — a successful neural network approach," 07 2014, pp. 3074–3077.

[7] Omer Sezer and Murat Ozbayoglu, "Algorithmic financial trading with deep convolutional neural networks: Time series to image conversion approach," *Applied Soft Computing*, vol. 70, 04 2018.

[8] Wei Bao, Jun Yue, and Yulei Rao, "A deep learning framework for financial time series using stacked autoencoders and long-short term memory," *PLOS ONE*, vol. 12, pp. 1–24, 07 2017.

[9] Weiwei Jiang, "Applications of deep learning in stock market prediction: recent progress," Papers, arXiv.org, Feb. 2020.

[10] C.B. Garcia and F.J. Gould, "Survivorship bias," *The Journal of Portfolio Management*, vol. 19, no. 3, pp. 52–56, 1993.

[11] M. Shaha and M. Pawar, "Transfer learning for image classification," in *2018 Second International Conference on Electronics, Communication and Aerospace Technology (ICECA)*, 2018, pp. 656–660.

[12] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson, "How transferable are features in deep neural networks?," in *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, Eds., pp. 3320–3328. Curran Associates, Inc., 2014.

[13] H. Ismail Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P. Muller, "Transfer learning for time series classification," in *2018 IEEE International Conference on Big Data (Big Data)*, 2018, pp. 1367–1376.

[14] Adriano Koshiyama, Sebastian Flennerhag, Stefano B. Blumberg, Nick Firoozye, and Philip Treleaven, "Quantnet: Transferring learning across systematic trading strategies," 2020.

[15] Brian Kenji Iwana and Seiichi Uchida, "An empirical survey of data augmentation for time series classification with neural networks," *arXiv preprint arXiv:2007.15951*, 2020.

[16] Q. Wen, Liang Sun, Xiaomin Song, J. Gao, X. Wang, and Huan Xu, "Time series data augmentation for deep learning: A survey," *ArXiv*, 2020.

[17] Terrance DeVries and Graham W. Taylor, "Dataset augmentation in feature space," 2017.

[18] Brian Kenji Iwana and Seiichi Uchida, "Time series data augmentation for neural networks by time warping with a discriminative teacher," in *2020 25th International Conference on Pattern Recognition (ICPR)*, 2020.

[19] Terry T. Um, Franz M. J. Pfister, Daniel Pichler, Satoshi Endo, Muriel Lang, Sandra Hirche, Urban Fietzek, and Dana Kulić, "Data augmentation of wearable sensor data for parkinson's disease monitoring using convolutional neural networks," in *Proceedings of the 19th ACM International Conference on Multimodal Interaction*, 2017, ICMI '17, p. 216–220.

[20] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., pp. 8024–8035. Curran Associates, Inc., 2019.

[21] Arthur Le Guennec, Simon Malinowski, and Romain Tavenard, "Data augmentation for time series classification using convolutional neural networks," in *ECML/PKDD Workshop on Advanced Analytics and Learning on Temporal Data*, 2016.

[22] C.R. Bacon, *Practical Risk-Adjusted Performance Measurement*, The Wiley Finance Series. Wiley, 2012.