Predicting the Performance of a Computing System with Deep Networks

Mehmet Cengiz
Newcastle University
Newcastle upon Tyne, UK
m.cengiz2@ncl.ac.uk

Matthew Forshaw
Newcastle University
Newcastle upon Tyne, UK
matthew.forshaw@ncl.ac.uk

Amir Atapour-Abarghouei
Durham University
Durham, UK
amir.atapour-abarghouei@durham.ac.uk

Andrew Stephen McGough
Newcastle University
Newcastle upon Tyne, UK
stephen.mcgough@newcastle.ac.uk

ABSTRACT

Predicting the performance and energy consumption of computing hardware is critical for many modern applications. This will inform procurement decisions, deployment decisions, and autonomic scaling. Existing approaches to understanding the performance of hardware largely focus around benchmarking – leveraging standardised workloads which seek to be representative of an end-user's needs. Two key challenges are present; benchmark workloads may not be representative of an end-user's workload, and benchmark scores are not easily obtained for all hardware. Within this paper, we demonstrate the potential to build Deep Learning models to predict benchmark scores for unseen hardware. We undertake our evaluation with the openly available SPEC 2017 benchmark results. We evaluate three different networks, one fully-connected network along with two Convolutional Neural Networks (one bespoke and one ResNet inspired) and demonstrate impressive R^2 scores of 0.96, 0.98 and 0.94 respectively.

CCS CONCEPTS

• **Computing methodologies** → **Neural networks**.

KEYWORDS

deep networks, benchmarking, performance

ACM Reference Format:
Mehmet Cengiz, Matthew Forshaw, Amir Atapour-Abarghouei, and Andrew Stephen McGough. 2023. Predicting the Performance of a Computing System with Deep Networks. In *Proceedings of the 2023 ACM/SPEC International Conference on Performance Engineering (ICPE '23), April 15–19, 2023, Coimbra, Portugal.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3578244.3583731

1 INTRODUCTION

Performance benchmarks are commonly used as a tool to better understand systems. This includes informing procurement decisions and, through the operation of systems, to inform deployment and scaling decisions. These benchmarks seek to understand the likely performance of a user's workload, but also energy consumption. While benchmarks show good potential to gain an understanding of performance, it is typically infeasible to benchmark all possible combinations of workload and hardware. This problem is exacerbated in environments which exhibit hardware heterogeneity.

Benchmarks [1, 5, 21] – which produce metrics [11, 19] on different hardware under specific workloads, help to identify the 'best' hardware. The metrics can then be compared for different hardware options, supporting judgements as to how a specific user's workload would be expected to perform.

We seek to resolve the challenge of evaluating the performance for previously unseen hardware-workload combinations, using the SPEC CPU 2017 dataset. Previous efforts using linear regression (e.g., [17]) have demonstrated the potential to predict performance metrics, but perform poorly for non-linear aspects of hardware evolution. In our work we present a data cleaning pipeline to ensure the data is amenable to modelling.

We explore the potential of three Deep Networks to better model non-linear relationships in the benchmark data. We evaluate a number of fully-connected networks (often referred to as multilayer perceptrons (MLP)) due to the tabular format of the dataset as well as Convolutional Neural Networks (CNN). Originally developed for learning from image-based data (2-dimensional, greyscale, or 3-dimensional, colour), CNNs have recently gained traction in the case of 1-dimensional datasets such as tables [3, 4, 30]. For the first CNN approach, we evaluate a number of networks which contain convolution and pooling operations whilst for the second CNN approach, we evaluate adding residual blocks as proposed in ResNet [10]. We perform a hyperparameter tuning process within each of these networks. This allows us to demonstrate our approach can accurately predict unseen benchmark results. From this we are able to achieve R^2 scores of 0.96, 0.98 and 0.94 respectively, compared to 0.53 for linear regression.

The remainder of this paper is organised as follows. In Section 2, we discuss prior work focusing on performance prediction. We outline our methodology in Section 3. We present our results in Section 4 and explore Threats to Validity in Section 5. We conclude and outline areas of future work in Section 6.

ICPE '23, April 15–19, 2023, Coimbra, Portugal

ACM ISBN 979-8-4007-0068-2/23/04.
https://doi.org/10.1145/3578244.3583731

Table 1: An overview of the prediction studies that used SPEC datasets.

Work	Dataset(s)	Technique(s)	Prediction
[17]	SPEC CPU / SPEC Java Server	Custom linear regression model	Server benchmark performances
[22]	SPEC 2006	Custom linear regression model	Performance of future systems
[8]	SPEC CPU2000 / CPU2006	Hybrid mechanistic-empirical model	Commercial processor performance
[13]	SPEC OpenMP	Classic fractal-based sampling	Accelerating multithreaded app simulation
[29]	SPEC 2006	Fine-grained phase-based approach	Performance and power
[20]	SPEC 2017	Multiple Neural Networks	Computer hardware configuration
[24]	SPEC 2017	Multi-layer perceptron	Computer performance
Ours	SPEC 2017	MLP, CNN	Computer performance

2 RELATED WORK

Here we present prior work on ML-based performance prediction of computer systems.

Performance prediction is the process of predicting some performance metric for a system based on known characteristics of that system, which is sometimes referred to as empirical performance modeling [26]. However, we will reduce the scope of our study here down to the prediction of performance metrics for computer systems. In general, performance prediction is for values which can take any value within a given range (e.g., time to complete some task or a numeric value used to compare different systems). As such, the work here focuses on regression techniques.

One of the earliest studies was performed by Ein-Dor and Feldmesser [6]. They claimed that by using readily available data on CPU characteristics, it is possible to predict a given CPU performance. However, their work is based around simple statistical approaches and cannot be used for the SPEC performance predictions we wish to perform here. Ipek *et al.* [12] used artificial neural networks to predict Instructions per Cycle (IPC) of a given system. Their dataset contains L1 and L2 cache sizes –the first and second caches in the hierarchy of cache levels– and front-side bus bandwidth. Their experiments showed that their model predicts IPC with only a 1-2% error.

Li *et al.* [18] carried the empirical performance prediction domain to the cloud environment by developing a tool named CloudProphet. This was effectively a trace-and-replay tool to predict a legacy application's performance if migrated to a cloud infrastructure. As our work here focuses on prediction of benchmark scores, this would not be easily translatable to our work, though it could form a good starting point for predicting the performance of a specific workload on another (non-cloud) computer.

Upadhyay *et al.* [25] discuss performance prediction issues from a different point of view. Their motivation is to consider the other components of a systems hardware while designing a CPU. For selecting the best combination of CPU, they used data mining techniques. Although this could be applied to the SPEC datasets we would argue that the non-linear nature of new hardware would make this a less than accurate approach.

A number of prediction approaches have been proposed for prediction of performance metrics for GPUs. Ardalani *et al.* [2] focused on GPU performances and designed an ensemble of regression learners named Cross-Architecture Performance Prediction (XAPP). However, they intended to predict GPU performances using single-threaded CPU implementations. They achieved a 26.9% average error on a set of 24 real-world kernels. As they mentioned in their paper, their study cannot capture the impact of texture memory and constant memory. On the other hand, adhering to their implication, this is the problem of having a small dataset that contains 122 data points. Therefore, since our dataset contains more than 20K data points, we require more sophisticated models.

The work by Justus *et al.* [14] forms inspiration for our work as they used Multi-Layer Perceptrons for the prediction of execution time for training Deep Learning networks. However, we take this work further by using Convolutional Neural Networks for our predictions and apply it to the SPEC dataset.

2.1 Predictions from the SPEC datasets

A number of works have addressed the problem of predicting metrics for the SPEC datasets. As there have only been two prior works which address the SPEC 2017 dataset, we expand our discussion here to cover all of the SPEC datasets. A summary of these works can be found in Table 1.

Lee [17] and Ozisikyilmaz *et al.* [22] used linear regression models for predicting benchmark performance. Our work seeks to overcome potential limitations by modelling non-linear responses.

Eyerman *et al.* [8] developed a mechanistic model built on interval analysis which breaks the total execution time into intervals based on missed events, for out-of-order superscalar processors.

Jiang *et al.*. [13] presented a study to evaluate design alternatives for computer architectures. They designed a fractal-based sampling to speed up parallel microarchitecture simulation with multithreaded applications. Due to the fact that they mainly intend to obtain samples from parallel programming datasets, the only similarity with our study is the use of SPEC-based datasets.

Zheng *et al.* [29] proposed a unified learning-based framework named LACross to estimate time-varying software performance and power consumption on a target hardware platform.

Lopez *et al.* [20] used multiple neural networks for a classification task for predicting the best computer hardware configuration options. Although their work demonstrates the validity of using Deep Learning on SPEC datasets, their underlying problem is quite different to ours. The closest work to ours is that of Tousi and Lujan [24], which uses MLPs for the prediction of computer performance. We go further by demonstrating how the use of Convolutional Neural Networks can be used to provide better results.

Table 2: Columns of SPEC2017

Data Type	Column
String	Benchmark, Hardware Vendor, System, Processor, CPU(s) Orderable, 1st Level Cache, 2nd Level Cache, 3rd Level Cache, Other Cache, Storage, Operating System, File System, Compiler, License, Tested By, Test Sponsor
Numerical	Peak Result, Base Result, Energy Peak Result, Energy Base Result, # Cores, # Chips, Memory, # Enabled Threads Per Core, Processor MHz
Binary	Parallel
Ternary	Base Pointer Size
Quaternary	Peak Pointer Size
Date (mon-yyyy)	HW Avail, SW Avail, Test Date, Published, Updated
Text	Disclosures

3 METHODOLOGY

All experiments are run on a Tesla T4 GPU and two Intel Xeon(R) CPUs @ 2.30GHz, and 12 GB of memory. As the SPEC 2017 dataset is not directly in a format which can be used for machine learning, we first discuss the process used for dataset cleansing in order to provide data which can be fed directly to our Deep Learning networks. We then go on to cover the search space of Deep Learning networks which we have evaluated as part of this work.

3.1 Dataset cleansing

Within this work we consider how to prepare SPEC 2017 benchmark dataset for machine learning. The dataset includes 34 attributes as illustrated in Table 2. The numeric columns *Peak Result* and *Base Result* represent the response time of systems under load or no load respectively and are the values we seek to predict in this work. We perform the following pre-processing on the data, making it amenable for model training. Our approach to mitigate inconsistencies and data quality issues include the following:

- **Alphanumeric cleaning:** Non-alphanumeric characters such as tabs and escape characters are removed from the dataset. We also remove spaces from column names to make downstream processing easier. All characters are converted to lower case to remove inconsistencies.
- **Removal of outliers:** Some of the *Base Result* values were zero, which is clearly incorrect. As there were only a small number of these, they are removed.
- **Making units consistent:** Units varied across the data (e.g., memory in KB, MB, GB). All units are standardised to MB.
- **Make columns categorical:** Many of the columns although appearing to allow arbitrary data are actually highly constrained (e.g., Memory can only take a small range of values). As such, the set of these values was determined and the data was replaced with categorical labels.
- **Removal of highly correlated columns:** We used Kendall's rank correlation [23] to identify those columns which are highly correlated. It was determined, in our case, that the columns 'CPU(s) Orderable', 'Energy Base Result', 'License', 'Parallel', 'System', 'Test Sponsor', and 'Tested By' were more than 70% correlated with other columns. As strongly correlated variables may have almost the same ability to predict the result value for observation, due to their linear dependence, they were eliminated. It should be noted that we also evaluated Pearson and Spearman correlation and obtained similar results.

3.2 Searching for the 'best' Neural Network

The shape (layers and neurons per layer) of Deep Learning networks significantly impact performance. We perform a space search for the 'best' network for the SPEC data. We identify three network structures, two trapezium and one rectangular and populate these with either single neurons, Convolutional nodes or Residual Nodes. We evaluate three core network designs within this work. Those of fully-connected networks, convolutional neural networks and networks which use Residual blocks as proposed by the ResNet architecture [10]. We detail the design of each of these architectures:

3.2.1 Fully-Connected Networks: We evaluate three network structures, those of a strictly decreasing number of neurons per layer shaped network – which we will refer to as a trapezium network hereafter, see Figure 1, the reverse of this – referred to as a reverse trapezium – and that of a rectangular network with the same number of neurons in each layer. For the trapezium network the first layer has 2^n neurons. Each subsequent layer has half the number of neurons as the previous layer. The penultimate layer has 2^{n-m} neurons where $n - m > 1$. We vary the values of n in the range $[4, ..., 11]$ and m in the range $[1, ..., 10]$. The final layer of the network contains just a single neuron to provide the regression result. Reverse trapezium networks flip the order of the layers (apart from the last) having the narrowest layer first and the widest layer last.

The rectangular networks contain m layers and have 2^n neurons in each layer, with a final layer containing only one neuron to provide the regression result. Although this network does not vary in shape between layers, the network learns weights which cause some neurons in a level to become redundant, effectively learning itself the number of neurons to place in each layer.

3.2.2 CNN design: The CNN network consists of a number of convolutional layers followed by a fully-connected set of layers. Figure 2 illustrates the shape of these networks. It should be noted that in these cases the fully-connected layers are smaller than those

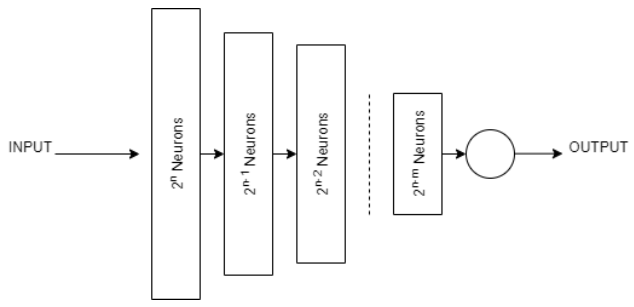

Figure 1: Sketch view of trapezium-shaped MLPs

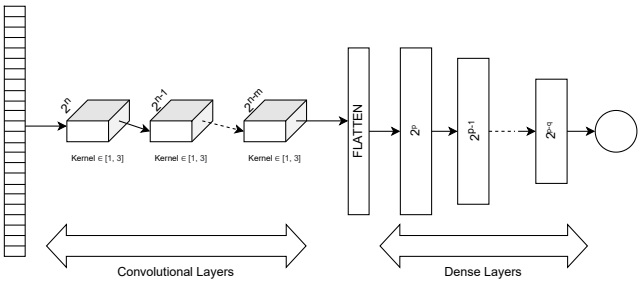

Figure 2: The CNN network structure

where we only use fully-connected layers. As our data is tabular, we use 1D convolutional layers – i.e. our kernels/filters are 1D and of size $k \in [2, ..., 5]$. Again, we adopt the trapezium format of the first convolutional layer having a width (number of filters) of 2^n and each subsequent layer having half the width of the previous layer. With the last convolutional layer having a width of 2^{n-m} ($n - m > 1$). The fully-connected layers are trapezium in shape and range in nodes per layer between 2^p and 2^{p-q}. We allow n, m, p and q to vary in the ranges $[7, ..., 11]$, $[4, ..., 7]$, $[7, ..., 11]$, and $[5, ..., 7]$, respectively. Initial experiments indicated that searches within these ranges yielded the best results.

3.2.3 Residual design: We adopt Residual blocks [10], where a 'bypass' link around a set of convolutional units is merged with the output from the convolutional units. Figure 3 illustrates this network topology and we refer to this hereafter as the identity block. The width of the input and output to the identity block must be the same (2^p). By convention, the kernel size of the first two convolutions are 2^{p-2} with the kernel size of the last convolution being 2^p to restore the original size. We allow $p \in [6, ..., 11]$.

One restriction of the original identity block is that the shape of the data entering the block must be the same as the shape of the output – otherwise the merging of the data from the 'bypass' will not be possible. In order to overcome this, we use a convolution unit to the 'bypass' which has the same output width as the final convolution in the main path – see Figure 4. In this case, the first two convolutions on the main path have a width of 2^{p-2}, while the last convolution on the main path and the 'bypass' path have widths of 2^p. We refer to this as a convolutional block.

The two block templates are then combined to produce a superblock (Figure 5). Each superblock starts with a convolutional block followed by r identity blocks. The width of the output for each block (both identity and convolution) within a block will be 2^p, also the output width of the whole superblock.

Superblocks can then be concatenated together as in Figure 6. Here, the original vector data is fed into a set of w superblocks.

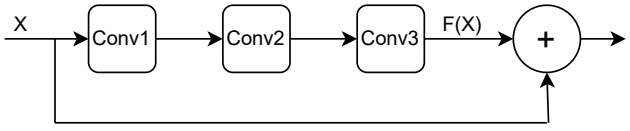

Figure 3: The identity block

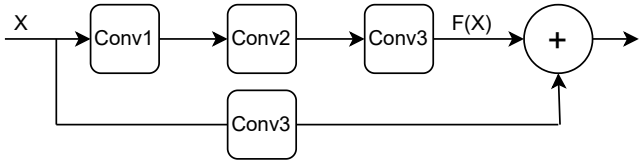

Figure 4: The convolutional block

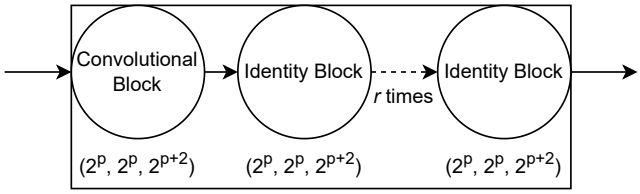

Figure 5: A superblock constructed from a convolution block and r identity blocks

Following the convention of ResNet, the width of output from each superblock will be double that of the previous superblock. Finally the output from the last superblock will be flattened before being fed into a single neuron to predict the regression value.

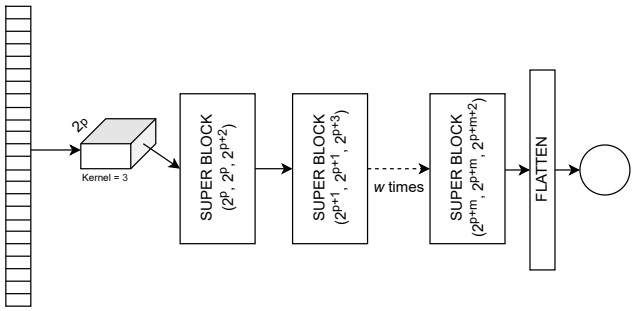

Figure 6: The ultimate design of our ResNet model

3.3 Hyperparameter search

In addition to performing an neural architecture search over the architecture range specified in 3.2, we also conducted a thorough search across the hyperparameters which could be used for the networks. This included the optimiser, the number of training epochs, the loss function and the activation function.

3.3.1 Optimiser: The optimiser is used to determine how the weights of the network are updated after each training step. This work focuses on three of the most commonly used optimisers:

SGD: Stochastic Gradient Decent is the original optimiser used for Deep Learning. Although strictly speaking Gradient decent performs an update after each training sample is processed, we adopt the normal convention of performing the optimisation step after each batch of data is processed – more correctly referred to as Batched Stochastic Gradient Decent.

RMSprop: Root Mean Squared Propagation extends SGD by applying decaying average partial gradients to the step size of each parameter. The optimiser focuses more on recent gradients.

Adam: ADAptive Moment estimation [16] is an extension of SGD. Like RMSprop, Adam adopts a separate learning rate for each parameter. While RMSprop uses the average of the first moment, Adam also uses the average of the second moment when choosing how to adapt the learning rates.

3.3.2 Loss function: The loss function is used to determine the difference between the predicted values and the true values. We evaluate two loss functions, Mean Squared Error (MSE, Equation 1) and Mean Absolute Error (MAE, Equation 2),

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i' - y_i)^2, \quad (1) \qquad MAE = \frac{1}{N}\sum_{i=1}^{N}|y_i' - y|, \quad (2)$$

where number of samples is N, y_i' and y_i are predicted and true values respectively. Larger errors will have a larger impact on MSE's loss value which we would expect to lead to fewer outlier values.

3.3.3 Activation function: The activation function provides the non-linear element within the networks. We evaluated three commonly used activation functions within our work: sigmoid, tanh and ReLU. For many problems ReLU has been shown to be the most effective activation function. However, there are a number of cases where the other activation functions are more suited to the problem at hand. As is the convention, no activation function was used on the final output layer to allow for arbitrary output values.

3.3.4 Stride: Stride is the number of cells that shifts over the input matrix. While stride size is adaptive in the CNN experiments, we kept it fixed in the residual-inspired models except for the starting layer of the convolutional blocks. Our dataset has 24 columns for independent variables, making the shape of input data (1, 24), we defined the stride size in the range of $[1, ..., 4]$.

3.4 Implementation Details

We use an 80-20 training-test split. Further, the training data is split into training and validation sets as 80% and 20% respectively. The batch size is determined as 10. Each model is trained 5 times with different random seeds to obtain its average performance. The data splitting process was purposely designed to demonstrate the real-world future unseen data in operation, and adjusting class distributions via any controlled split approach such as cross-validation would go against an uncontrolled future configuration of data [28]. By using random seeds to generate random splits of data for each experiment, we can be confident that our models are capable of responding to random distributions of data. in the real world.

We use the Glorot uniform initialiser [9] for initialising the parameters within our networks, which sets the weights so they are equal across all layers in terms of the variance of the activations. The gradient is kept from exploding or vanishing by the constant variance. In addition, the initial bias value(s) were set to zero [15].

We allowed the number of epochs to vary between 100 and 300 in steps of 50. We stopped training after 300 epochs, where models demonstrated optimal performance on the test set.

3.4.1 Baseline Models. We consider three baseline models:

Linear Regression: We would expect that this model would perform well for similar hardware, but perform poorly when there is a non-linear change in hardware performance.

Support Vector Regression: SVR is often better than a linear regression model as it is able to fit better to the model. However, it still suffers from the fact that it is a linear model and hence is not expected to adapt well to step-changes in the hardware.

Random Forest Regression: This is an ensemble technique which does not suffer from the linear model problems of the other two approaches. It does, however, require prior examples of hardware types to be able to predict new hardware accurately. We would therefore expect this to be better than the other baseline models, but less likely to be adaptable as the Deep Learning models.

3.4.2 Evaluation Metrics: We evaluate model performance using MSE (Equation 1), MAE (Equation 2) and R^2:

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

where y_i is the true value, $\hat{y}_i$ is the predicted value and $\bar{y}$ is the mean of all true values. For both the R^2 and MSE, values further from their predicted value are going to have a more significant impact on the results. In order to measure the models' vulnerability to outliers, focusing on MSE would be preferable. A focus on R^2 would reduce outliers at the expense of overall accuracy.

4 RESULTS

We present the results of our model training. All results represent the average of the five different splits of the dataset. Tables 3 and 4 present the top performing models when sorted by R^2 and MSE.

4.1 Baseline Models

We first evaluate our baseline cases. Both Linear Regression ($R^2 = 0.526$, $MSE = 15761.2$) and Support Vector Regression ($R^2 = -0.004$, $MSE = 33448.31$) performed poorly. Figure 7 shows Quantile-Quantile plots for the residuals of the top-performing model of each type; CNN, Linear Regression, Random Forest and SVR. We observe that the CNN models exhibit preferable behaviour at both extremes. Meanwhile, linear regression and Random Forest models exhibit larger residuals for lower quantiles. Finally, Linear Regression, Random Forest and SVR exhibit large variances for high performance machines in the dataset. Figure 8 shows the magnitude of residuals for each methods. We observe our CNN based approach exhibits preferable behaviour to prior approaches.

4.2 Deep Learning Models

For MLP networks, the Trapezium networks offered highest performance, achieving 45th position for R^2 and 48th position for MSE. The performance of MLP networks were typically not competitive with CNN-based approaches, so we do not discuss them further. Meanwhile, CNN networks dominate the top 12 and 13 positions for R^2 and MSE respectively.

We hypothesised that residual-inspired approaches would perform favourably in our case, due to their strong performance in other domains, however, this is not borne in our findings. Residual-inspired approaches only acheived positions of 159th by R^2 and

Table 3: The results of the best deep networks and machine learning models – Order of R^2

#	Architecture	Loss Fn	Kernel Sizes	Stride Sizes	Number of Filters (m, n)	Neurons in Layers (p, q)	Optimizer	Epochs	R2	MAE	MSE
1	TriCNN	MAE	3	1	(9, 7)	[9, …, 5]	Adam	250	0.98638701	5.67389728	465.3285655
2	TriCNN	MAE	3	2	(9, 7, 6, 5, 4)	[9, …, 4]	Adam	250	0.98590661	5.83946465	476.0394343
3	TriCNN	MAE	3	2	(9, 7)	[9, …, 5]	Adam	300	0.98579341	5.76197731	494.124225
4	TriCNN	MAE	3	1	(9, 7)	[9, …, 5]	Adam	150	0.98529142	6.25318407	513.9629513
5	TriCNN	MAE	3	2	(9, 7, 6, 5, 4)	[9, …, 4]	RmsProp	150	0.98282719	7.14056732	620.2982421
6	TriCNN	MAE	3	2	(9, 7, 6, 5, 4)	[9, …, 4]	Adam	200	0.98280914	6.03564805	582.3068145
7	TriCNN	MAE	3	2	(9, 7, 6, 5, 4)	[9, …, 4]	Adam	300	0.98278342	5.61076184	582.0247239
8	TriCNN	MAE	3	1	(9, 7)	[9, …, 5]	Adam	300	0.98107176	5.78137347	645.4129883
9	TriCNN	MAE	3	2	(9, 7)	[9, …, 5]	RmsProp	250	0.98095925	6.72097815	669.8856237
10	TriCNN	MAE	3	1	(9, 7)	[9, …, 5]	Adam	200	0.98089907	6.32291809	665.1641919
11	TriCNN	MAE	3	2	(9, 7)	[9, …, 5]	Adam	150	0.98047251	6.71537772	663.7030719
12	TriCNN	MAE	3	1	(7, 6, 5, 4)	[9, …, 5]	RmsProp	300	0.98038864	6.9974749	653.5821786
~	RF								0.9803076	4.76701531	688.0001262
13	TriCNN	MAE	3	1	(7, 6, 5, 4)	[9, …, 5]	RmsProp	200	0.98002879	7.62788323	684.7595471
14	TriCNN	MAE	2	1	(9, 7)	[11, …, 6]	Adam	150	0.9793459	6.519971	703.0615545
15	TriCNN	MAE	3	2	(9, 7)	[9, …, 5]	Adam	100	0.97782539	8.23651529	754.5381605
16	TriCNN	MAE	3	2	(9, 7, 6, 5, 4)	[9, …, 4]	Adam	100	0.97748578	7.30871799	757.4994833
17	TriCNN	MAE	3	2	(9, 7, 6, 5, 4)	[9, …, 4]	Adam	150	0.97726148	6.65855022	772.0747562
18	TriCNN	MAE	3	1	(7, 6, 5, 4)	[9, …, 5]	RmsProp	250	0.97665471	7.86703389	775.8960386
19	TriCNN	MAE	3	2	(9, 7, 6, 5, 4)	[9, …, 4]	RmsProp	250	0.97650919	7.97325412	852.3545636
20	TriCNN	MAE	3	2	(9, 7)	[9, …, 5]	RmsProp	300	0.97636563	6.91501173	816.7881606
45	TriMLP	MAE				[11, …, 6]	Adam	250	0.97347275	9.12443258	906.1439402
159	Residual	MAE	Number of Superblocks = (2, 5, 5, 2)		((6, 6, 8), (7, 7, 9), (8, 8, 10), (9, 9, 11))	1	RmsProp	250	0.95007233	10.595069	1006.134564
~	LR								0.52639158	82.4596122	15761.16107
~	SVR								-0.0045634	113.749207	33448.30886

* TriCNN = Trapezium-shaped CNN, RF = Random Forest Regression, TriMLP = Trapezium-shaped MPL, LR = Linear Regression, SVR = Support Vector Regression

Table 4: The results of the best deep networks and machine learning models – Order of MSE

#	Architecture	Loss Fn	Kernel Sizes	Stride Sizes	Number of Filters (m, n)	Neurons in Layers (p, q)	Optimizer	Epochs	R2	MAE	MSE
1	TriCNN	MAE	3	1	(9, 7)	[9, …, 5]	Adam	250	0.98638701	5.67389728	465.3285655
2	TriCNN	MAE	3	2	(9, 7, 6, 5, 4)	[9, …, 4]	Adam	250	0.98590661	5.83946465	476.0394343
3	TriCNN	MAE	3	2	(9, 7)	[9, …, 5]	Adam	300	0.98579341	5.76197731	494.124225
4	TriCNN	MAE	3	1	(9, 7)	[9, …, 5]	Adam	150	0.98529142	6.25318407	513.9629513
5	TriCNN	MAE	3	2	(9, 7, 6, 5, 4)	[9, …, 4]	Adam	300	0.98278342	5.61076184	582.0247239
6	TriCNN	MAE	3	2	(9, 7, 6, 5, 4)	[9, …, 4]	Adam	200	0.98280914	6.03564805	582.3068145
7	TriCNN	MAE	3	2	(9, 7, 6, 5, 4)	[9, …, 4]	RmsProp	150	0.98282719	7.14056732	620.2982421
8	TriCNN	MAE	3	1	(9, 7)	[9, …, 5]	Adam	300	0.98107176	5.78137347	645.4129883
9	TriCNN	MAE	3	1	(7, 6, 5, 4)	[9, …, 5]	RmsProp	300	0.98038864	6.9974749	653.5821786
10	TriCNN	MAE	3	2	(9, 7)	[9, …, 5]	Adam	150	0.98047251	6.71537772	663.7030719
11	TriCNN	MAE	3	1	(9, 7)	[9, …, 5]	Adam	200	0.98089907	6.32291809	665.1641919
12	TriCNN	MAE	3	2	(9, 7)	[9, …, 5]	RmsProp	250	0.98095925	6.72097815	669.8856237
13	TriCNN	MAE	3	1	(7, 6, 5, 4)	[9, …, 5]	RmsProp	200	0.98002879	7.62788323	684.7595471
~	RF								0.9803076	4.76701531	688.0001262
14	TriCNN	MAE	2	1	(9, 7)	[11, …, 6]	Adam	150	0.9793459	6.519971	703.0615545
15	TriCNN	MAE	3	2	(9, 7)	[9, …, 5]	Adam	100	0.97782539	8.23651529	754.5381605
16	TriCNN	MAE	3	2	(9, 7, 6, 5, 4)	[9, …, 4]	Adam	100	0.97748578	7.30871799	757.4994833
17	TriCNN	MAE	3	2	(9, 7, 6, 5, 4)	[9, …, 4]	Adam	150	0.97726148	6.65855022	772.0747562
18	TriCNN	MAE	3	1	(7, 6, 5, 4)	[9, …, 5]	RmsProp	250	0.97665471	7.86703389	775.8960386
19	TriCNN	MAE	3	2	(9, 7)	[9, …, 5]	RmsProp	200	0.97613855	7.72461632	807.1294185
20	TriCNN	MAE	3	2	(9, 7)	[9, …, 5]	RmsProp	300	0.97636563	6.91501173	816.7881606
48	TriMLP	MAE				[11, …, 6]	Adam	250	0.97347275	9.12443258	906.1439402
135	Residual	MAE	Number of Superblocks = (2, 5, 5, 2)		((6, 6, 8), (7, 7, 9), (8, 8, 10), (9, 9, 11))	1	RmsProp	250	0.95007233	10.595069	1006.134564
39	LR								0.52639158	82.4596122	15761.16107
40	SVR								-0.0045634	113.749207	33448.30886

* TriCNN = Trapezium-shaped CNN, RF = Random Forest Regression, TriMLP = Trapezium-shaped MPL, LR = Linear Regression, SVR = Support Vector Regression

135th by MSE. These models are more complex to engineer, and require more time to train, and provided little benefit in our case.

We now summarise other design choices:

Optimizer: Consistent with prior research, Adam generally performed best, though RMSprop is a strong contender.

Loss function: In all cases MAE produced the best results. Somewhat surprising when the overall metric is MSE.

Activation Function: Sigmoid produced results for the smaller architectures; however, the results were either NaN or negative. On the other hand, the results of the tanh activation function could not pass 0.01 in terms of R2.

Stride Size: In most cases (R^2 and MSE) having a stride size of one and two are best.

Kernel Size: The best kernel size is three for the top result though this is not consistent over all results.

Training Epochs: Figure 9 shows the impact of training epochs on performance for our top model, measured by MAE, MSE and R^2. The epoch count for stopping our training was determined empirically.

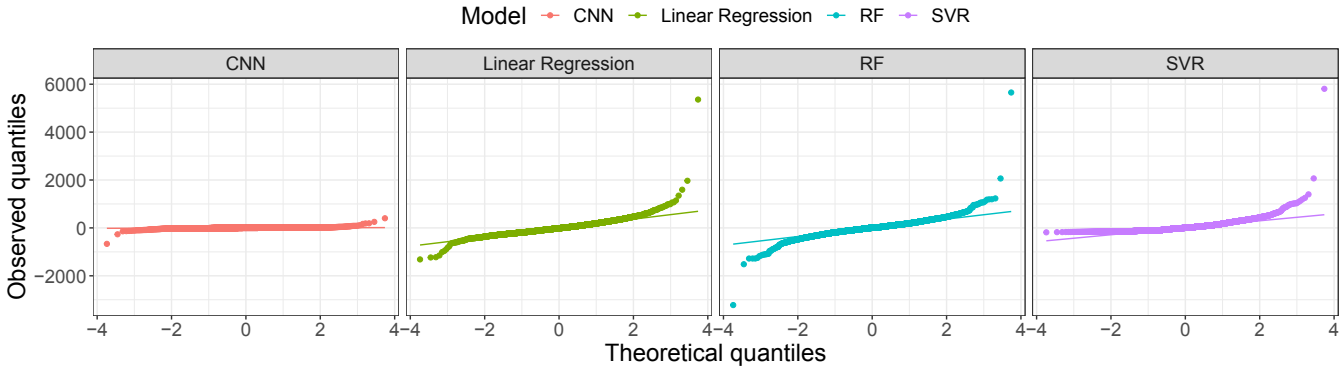

Figure 7: Q-Q Plots for residuals of the best performing CNN, Linear Regression, Random Forest and SVM models

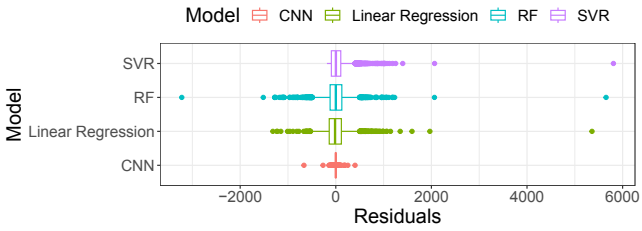

Figure 8: Magnitude of residuals, by method.

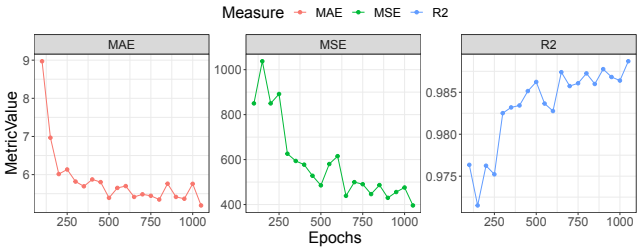

Figure 9: Performance for top CNN model, by training epoch.

4.3 Evaluation of top cases

Here, we evaluate the performance of the top four networks by R^2 and MSE. As seen in Table 3 and 4, four models which are trapezium-shaped CNNs that are optimised by Adam dominate both tables. All cases of the first-ranked model are in the top ten - 1st, 4th, 8th, and 10th in Table 3 and 1st, 4th, 8th, and 11th in Table 4. However, when the table is examined closely, we see that the increase in the number of epochs does not result positively for this model and a fluctuation in R^2 performance is observed. At this point, it is important to compare other metrics. If we sort all epoch setups of the first-ranked model by MAE, we see similar fluctuations in performance. However, considering the MSE values, although there is no significant difference between each model, the first-ranked configuration seems to be more resistant to outliers. As a result, it is expected that an increase/decrease in the number of epochs affects the performance increase/decrease, but counter-intuitively, no pattern is obtained in this case. Moreover, the same argument is valid for the second-ranked model in both tables. Furthermore, an increase in the quantity of convolutional layers does not always translate into an improvement in performance.

5 THREATS TO VALIDITY

Here, we introduce the limitations of this work, and highlight threats to validity arising from these. We structure our approach based on similar initiatives in the systems performance literature (e.g., [7]) and the approach of Wohlin *et al.* [27].

L1 Single benchmark dataset This study uses only data from SPEC CPU 2017 retrieved on 10 September 2022.

L2 Single expert for data cleaning Data cleaning processes were developed by a single expert researcher.

We now consider the implication of these limitations in terms of *construct*, *internal* and *external* validity.

Construct Validity This work concerns the prediction of performance results. Further work could have also evaluate whether predictive performance holds for columns *Energy Peak Result* and *Energy Base Result* from the dataset.

Internal Validity As highlighted in Section 3.1, our work involved cleaning data for it to be amenable to analysis and machine learning. The development of the cleaning processes were undertaken by a single expert researcher (**Limitation L2**), leaving the opportunity for misinterpretation of the datasets. To mitigate this impact, the processes undertaken were well documented, and the process was audited by two further researchers. Code to automate data cleaning is made available to the community.

External Validity Our experiment considers data from a single benchmark, SPEC CPU 2017 (**Limitation L1**), which may limit the generalisability of our findings. While our experiments were conducted for just one benchmark, our methodology is applicable to performance benchmarks more broadly. Further research is required to understand the extent to which our methods are effective for other workloads; we make this possible by providing our data and models for reproduction by other researchers.

Reproducibility We have made all our code and data, including the results of the training of all the networks available[1].

[1]https://github.com/cengizmehmet/BenchmarkNets

6 CONCLUSION

This work has considered the extent to which it is possibly to predict benchmark results for previously untested hardware configurations. We have specifically focused on the potential of using Deep Network approaches to capture the non-linear relationships present in the data. Our study has centred around the SPEC CPU 2017 dataset.

We investigated three deep network types, MLPs, CNNs, and an architecture of CNNs which is ResNet. After comprehensive studies, the models we offer excel at predicting the performance of a given system. While the R^2 values are between approx. 0.945 and 0.985, MAEs are between approx. 13 and 3.2. Secondly, it is discovered that convolutional layers can more efficiently predict our tabular data. This can be seen by examining the performance gain observed when adding convolutional layers to MLPs. Another finding of our paper involves demonstrating the effectiveness of residual blocks as opposed to simple convolutional layers. Our results indicate that while increasing the number of convolutional layers can offer promising results, the use of residual blocks leads to better performance overall.

This study is an indication and a starting point that deep neural networks can be trained on existing benchmark datasets to predict performance. However, we believe there are of many areas of future work. One avenue of future research would be to extend the application by taking advantage of more powerful neural network architectures with innovative feature aggregating modules or perhaps a higher parameter and layer count. Another direction of research would include exploring the effects of transfer learning, whereby the performance prediction system can be pre-trained on a larger proxy dataset to boost the performance after a subsequent and carefully designed fine-tuning process on the benchmark dataset. The use of synthetic data along with domain adaptation techniques can also lead to better performance and possibly steer the abilities of the model towards the desired outcome considering real-world data distributions. The method of procedurally generating the synthetic data in a meaningful manner that can benefit the training of neural network, perhaps in an end-to-end fashion, can also be an interesting area to investigate in a future work.

REFERENCES

[1] Gurumurthy Anand and Rambabu Kodali. 2008. Benchmarking the benchmarking models. *Benchmarking: An international journal* 15, 3 (2008), 257–291. https://doi.org/10.1108/14635770810876593

[2] Newsha Ardalani, Clint Lestourgeon, Karthikeyan Sankaralingam, and Xiaojin Zhu. 2015. Cross-Architecture Performance Prediction (XAPP) Using CPU Code to Predict GPU Performance. In *Proceedings of the 48th International Symposium on Microarchitecture* (Waikiki, Hawaii) *(MICRO-48)*. Association for Computing Machinery, New York, NY, USA, 725–737. https://doi.org/10.1145/2830772.2830780

[3] baosenguo. 2020. Mechanisms of Action (MoA) Prediction: 2nd Place Solution - with 1D-CNN.

[4] Ljubomir Buturović and Dejan Miljković. 2020. A novel method for classification of tabular data using convolutional neural networks. *bioRxiv* (2020). https://doi.org/10.1101/2020.05.02.074203

[5] Robert C Camp. 2006. *Benchmarking: the search for industry best practices that lead to superior performance* (1st. ed.). Productivity Press, NY.

[6] Phillip Ein-Dor and Jacob Feldmesser. 1987. Attributes of the Performance of Central Processing Units: A Relative Performance Prediction Model. *Commun. ACM* 30, 4 (apr 1987), 308–317. https://doi.org/10.1145/32232.32234

[7] Simon Eismann, Diego Elias Costa, Lizhi Liao, Cor-Paul Bezemer, Weiyi Shang, André van Hoorn, and Samuel Kounev. 2022. A case study on the stability of performance tests for serverless applications. *Journal of Systems and Software* 189 (2022), 111294.

[8] Stijn Eyerman, Lieven Eeckhout, Tejas Karkhanis, and James E. Smith. 2009. A Mechanistic Performance Model for Superscalar Out-of-Order Processors. *ACM Trans. Comput. Syst.* 27, 2, Article 3 (may 2009), 37 pages. https://doi.org/10.1145/1534909.1534910

[9] Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (Proceedings of Machine Learning Research, Vol. 9)*, Yee Whye Teh and Mike Titterington (Eds.). PMLR, Chia Laguna Resort, Sardinia, Italy, 249–256.

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep Residual Learning for Image Recognition. *CoRR* abs/1512.03385 (2015). arXiv:1512.03385 http://arxiv.org/abs/1512.03385

[11] Michelle M. Hugue. 2022. Lecture notes in Computer Systems Architecture.

[12] Engin Ïpek, Sally A. McKee, Rich Caruana, Bronis R. de Supinski, and Martin Schulz. 2006. Efficiently Exploring Architectural Design Spaces via Predictive Modeling. In *Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems* (San Jose, California, USA) *(ASPLOS XII)*. Association for Computing Machinery, New York, NY, USA, 195–206. https://doi.org/10.1145/1168857.1168882

[13] Chuntao Jiang, Zhibin Yu, Hai Jin, Chengzhong Xu, Lieven Eeckhout, Wim Heirman, Trevor E. Carlson, and Xiaofei Liao. 2013. PCantorSim: Accelerating Parallel Architecture Simulation through Fractal-Based Sampling. *ACM Trans. Archit. Code Optim.* 10, 4, Article 49 (dec 2013). https://doi.org/10.1145/2541228.2555305

[14] Daniel Justus, John Brennan, Stephen Bonner, and Andrew Stephen McGough. 2018. Predicting the Computational Cost of Deep Learning Models. In *2018 IEEE International Conference on Big Data (Big Data)*. 3873–3882. https://doi.org/10.1109/BigData.2018.8622396

[15] Keras. 2022. *Layer weight initializers*. https://keras.io/api/layers/initializers/ Last accessed 15 August 2022.

[16] Diederik P. Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. https://doi.org/10.48550/ARXIV.1412.6980

[17] Benjamin Lee. 2006. An architectural assessment of SPEC CPU benchmark relevance. *Harvard University, Cambridge, MA, Tech. Rep. TR-02-06* (2006).

[18] Ang Li, Xuanran Zong, Srikanth Kandula, Xiaowei Yang, and Ming Zhang. 2011. CloudProphet: Towards Application Performance Prediction in Cloud *(SIGCOMM '11)*. Association for Computing Machinery, New York, NY, USA, 426–427. https://doi.org/10.1145/2018436.2018502

[19] David J. Lilja. 2000. Measuring computer performance: A practitioner's guide.

[20] Leonardo Lopez, Michael Guynn, and Meiliu Lu. 2018. Predicting Computer Performance Based on Hardware Configuration Using Multiple Neural Networks. In *ICMLA*. 824–827. https://doi.org/10.1109/ICMLA.2018.00132

[21] Jean-Luc Maire, Vincent, Pillet Bronet, and Maurice Pillet. 2005. A typology of "best practices" for a benchmarking process. *Benchmarking: An international journal* 12, 1 (2005), 45–60. https://doi.org/10.1108/14635770510582907

[22] Berkin Ozisikyilmaz, Gokhan Memik, and Alok Choudhary. 2008. Machine Learning Models to Predict Performance of Computer System Design Alternatives. In *2008 37th International Conference on Parallel Processing*. 495–502. https://doi.org/10.1109/ICPP.2008.36

[23] David Sarmento. 2022. *Chapter 22: Correlation Types and When to Use Them.*

[24] Ashkan Tousi and Mikel Luján. 2022. Comparative Analysis of Machine Learning Models for Performance Prediction of the SPEC Benchmarks. *IEEE Access* 10 (2022), 11994–12011. https://doi.org/10.1109/ACCESS.2022.3142240

[25] Navin Mani Upadhyay, Ravi Shankar Singh, and Shri Prakash Dwivedi. 2022. Prediction of multicore CPU performance through parallel data mining on public datasets. *Displays* 71 (2022), 102112. https://doi.org/10.1016/j.displa.2021.102112

[26] Sam Van den Steen, Sander De Pestel, Moncef Mechri, Stijn Eyerman, Trevor Carlson, David Black-Schaffer, Erik Hagersten, and Lieven Eeckhout. 2015. Micro-architecture independent analytical processor performance and power modeling. In *2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. 32–41. https://doi.org/10.1109/ISPASS.2015.7095782

[27] Claes Wohlin, Per Runeson, Martin Höst, Magnus C Ohlsson, Björn Regnell, and Anders Wesslén. 2012. *Experimentation in software engineering*. Springer Science & Business Media.

[28] Xinchuan Zeng and Tony R. Martinez. 2000. Distribution-balanced stratified cross-validation for accuracy estimation. *Journal of Experimental & Theoretical Artificial Intelligence* 12, 1 (2000), 1–12. https://doi.org/10.1080/095281300146272

[29] Xinnian Zheng, Lizy K. John, and Andreas Gerstlauer. 2016. Accurate phase-level cross-platform power and performance estimation. In *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*. 1–6. https://doi.org/10.1145/2897937.2897977

[30] Yitan Zhu, Thomas Brettin, Fangfang Xia, Alexander Partin, Maulik Shukla, Hyunseung Yoo, Yvonne A. Evrard, James H. Doroshow, and Rick L. Stevens. 2021. Converting tabular data into images for deep learning with convolutional neural networks. *Scientific Reports* 11, 1, Article 11325 (2021), 11 pages. https://doi.org/10.1038/s41598-021-90923-y