Training time estimation for Convolutional Neural Network in Distributed Systems

Marco Schouten
EEMCS department
TU Delft
Delft, Netherlands
m.schouten-4@student.tudelft.nl
student number 5352908

Sayak Mukherjee
EEMCS department
TU Delft
Delft, Netherlands
s.mukherjee-7@student.tudelft.nl
student number 5222273

Abstract—**Training Deep Learning models is a resource-intensive task owing to the large parameter space and training data sets. Therefore training such models on a single computing unit takes a substantial amount of time. In recent years, distributed systems have been proposed to address this issue. Most cloud computing environments exploit distributed learning approaches to bring down training time drastically. However, this increase costs for the end-user. In this research, we have implemented a system that estimates the training time for a given deep learning model and optimizes system settings and hyper-parameters in relation to training time. Code is available at https://github.com/SayakMukherjee/fltk-testbed**

Index Terms—**Deep Learning, Distributed Systems, system and hyper-parameter optimization, Resource Management**

I. Introduction

A. Problem statement

Training Deep Learning on distributed cloud systems has been the subject of extensive research in the last few years to reduce training time by accessing computational power that would not be accessible otherwise to the end-user.

Most cloud computing environments exploit distributed learning approaches to bring down training time drastically. However, this requires a considerable amount of computing resources. Every computing resource accounts for additional expenses to the end-user. Additionally, state-of-the-art network architecture like Transformers [1] and Convolutional Neural Nets (CNN) [2] are built ad-hoc to be trained on distributed systems yielding more significant improvements in accuracy performance. In our case study, we will build a system that optimizes training time for CNN on a distributed system (Google Cloud).

B. Motivation

Cloud computing environments offer a variety of options as hardware resources. Therefore, making the optimal choice is a highly complicated task. There are often trade-offs between performance and associated costs when using such distributed cloud platforms for training deep learning models. A recent study proposed a Machine Learning model to choose the best cluster setup based on the user choosing to optimize for performance, cost, or both [4]. However, this implementation adds additional overhead.

C. Contribution

Our contribution is three fold:

- Design of Experiment (DoE) can be a powerful tool to find the optimal system settings given a hyperparameter setting without adding the overhead. This study explored a DoE approach to analyze the critical factors that influence training time and suggest the optimal combination of computing resources
- Analyze with Queuing Theory how a node with a single high-performance CPU compares to multiple parallel low-performance CPUs for training CNN.
- Build a regression model that predicts the training time for a given configuration using the DoE and Random Forest. Followed by using the model to predict maximum epochs that can be executed by a user given a time constraint

II. Background

The astounding success of Deep Nets in various application domains has led researchers to study the limitation and scalability of parallel training on distributed systems. Researchers adapted the base version of Stochastic Gradient Descent (the optimization algorithm) to a Parallelizing SGD. First, the global batch is divided into smaller chunks of equal size processed by n parallel workers, each containing a local copy of the model current parameters. Next, the gradients computed by each worker accumulated before the weight updates.

$$w_{t+1} = w_t + \eta \nabla x(w)$$

In [3] they show the limitations of standard multi computational node distributed systems for an overhead due to communication between nodes. This happens after only scaling to 4 or 8 nodes.

In a nutshell, for each iteration update, the current weights must be the same between all computational nodes (thus synchronizing model state). Additionally, Δw_{t+1} can be computed after w is available in every node. However, this communication must be done within the time of the next iteration. For fast nodes, communication time is longer than computation time, which is a limitation.

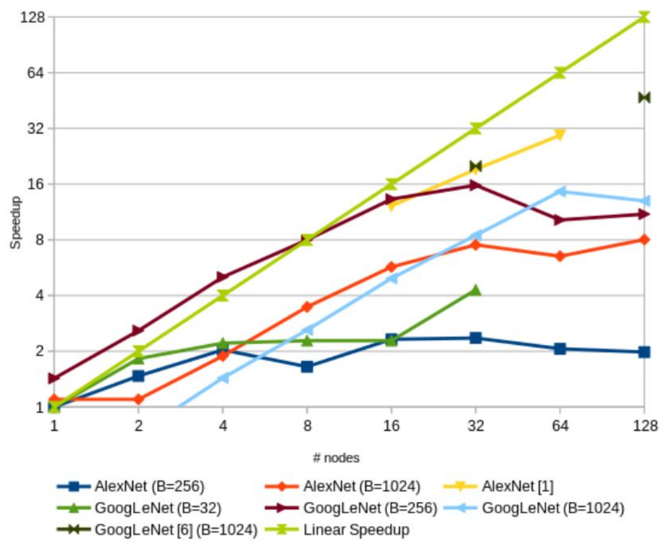


Fig. 1. DNN scalability tested on different batch sizes. [3] provided the speedup results in function of the number of computational nodes

Recently, google proposed *DistBelief* [4] a framework for distributed deep learning training. They proposed an asynchronous variant of SGD that reduces bandwidth to permit even more computational nodes to work in parallel. The key idea is to have a coordinator who has no direct access to the parameters. The coordinator's job is to send operations to each node, which keeps the results local. They have no overhead of sending parameters back and forth to a central server. In traditional distributed learning schemes, usually, the data is distributed among the various nodes, which then have the task of evaluating the gradient on a specific subset of data. And then send the gradients back to a central server. Instead, they proposed a load balancing scheme to solve this issue: the coordinator assigns small portions of work to each node and assigns new ones as soon as they run out. Thus faster models work more than slower. The coordinator then sends multiple copies to the various model replicas and uses the one that finishes first. To sum up, the parameters are sent at the beginning of each batch, and the gradients are sent every small amount of completed portions.

So as for traditional distributed learning schemes rely on an aggregator that collects the gradients of each node, [5] proposed an innovative method to reduce communication overhead by adding data compression modules within the network. It was possible because gradients are tolerant to approximations and precision loss. Thus they built a lossy-compression algorithm exploiting the loss-robustness property of gradients. Additionally, to avoid the compressors behaving as a bottleneck, they implemented a new learning algorithm that works without an aggregator (i.e. workers exchange gradients between themselves).

Another relevant study [6] over which our research takes inspiration is the development of Habitat: an innovative tool that helps to choose the most suitable GPU for training a deep net according to the user's needs. These recommendations are essential because hardware accelerators are expensive, and the user may prefer to be informed about costs and performances to make a better choice according to his or her needs. Habitat suggests these recommendations by predicting computational time and costs for GPUs which are not available; so the user can choose which one to rent.

It is possible because computations in DNN are highly repetitive, thus making predictions feasible.

Predictions depend on the chosen DNN architecture, hardware considered and software libraries used during training (e.g., cuDNN [74], cuBLAS [77]).

Habitat workflow operates in three steps: (1) Profile experimental data on an available GPU, (2) compute prediction using "Wave Scaling" or "Multi-Layer Perceptrons", (3) evaluate overall training execution time.

At the core, predictions for non-available GPUs depend on three scaling factors: Memory Bandwidth, wave size, clock frequency.

Lastly, they proposed two case studies that ranked a few GPUs according to cost and performances tuned for specific DNN models.

Their research addresses a problem of critical importance and provides a tool that offers quantitative suggestions. It is a massive step in this direction as current methods rely on heuristics which can have errors in predictions up to 49%, additionally getting measurements are tedious for different models, and this tool solves both problems.

III. Predictive Models

DNN training time prediction is vital as training on cloud services often leads to high monetary costs. In this research, we have analyzed training time using three different modelling techniques: DoE, Queuing Theory, Machine Learning.

A. Design of Experiments

Our first predictive model is based on DoE 2-level experimental design. Next, we investigate and quantify the relevance of factors that influence response time for a CNN training job. Our study considered 5 different factors: executor cores, parallelization coefficient, batch size, learning rate, max epoch. Reason of choice of each factor:

- Executor cores tell how many cores does the job need to be scheduled on. It predominantly affects the training time as higher computational power reduce the time it takes to process the input data. However, this has to be carefully tuned with the parallelization coefficient to get the best synchronicity and avoid a bottleneck.
- The parallelization coefficient determines how many parts the job should be split in. It is a critical component of distributed learning and affects the overhead of communication between nodes, as discussed in the background section

- Batch size is the critical component that enabled the development of Deep Nets through Batch Normalization [7]. The idea is to partition input data into smaller chunks and give one chunk at a time for training. It enabled a higher learning rate (thus reducing training time) and made the initial weights initialization less critical. Batch size refers to the number of data points selected to be normalized (mean and variance) before being given as input for backprop training with SGD. This factor has relevance and reduces training time by reaching faster convergence.
- Max epoch determines the limit of how many times the training algorithm goes through each data point. Therefore, it predominantly affects training time. One epoch means that each point in the dataset is considered for training. Multiple epochs mean that each point is considered multiple times. If the net reaches convergence before the max epoch (after checking the cross-entropy loss [8]), the training is usually stopped by using an Early Stopper.
- Learning rate is the most crucial hyper-parameter for Deep Learning Neural Nets. Numerous studies have been conducted on choosing the best strategy to initialize this value and decrease it over time. In our modelling, we are using Adam (Adaptive Moment Estimation). Adding momentum decreases training time by reducing the oscillations and variance of the model's parameters. The intuition here is that momentum keeps it moving in the previous direction, hence being more robust [1]

$$\theta_t = \theta_{t-1} - \alpha m_t / \sqrt{v_t} + \epsilon$$

This decaying value influence largely the convergence and accuracy, hence has impact on the training time

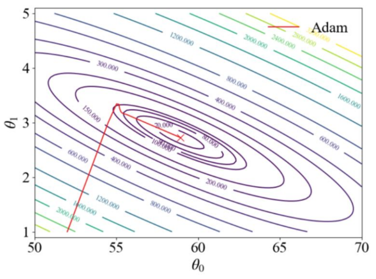


Fig. 2. Contour plots of Adam optimizer

B. Queuing Theory

We are analyzing two scenarios with Queuing Theory, intending to infer which yields the lowest response time - multiple low-performance nodes or single high-performance node.

[1]θ_t are next step model parameters, α is a coefficient internal of the algorithm, m_t and v_t are respectively first (mean) and second (uncentered variance) order moments of the gradient of the loss function (cross-entropy), ϵ is a small number that helps with stability

For this case, we are considering M/G/k and M/G/1 queues. Specifically, we are setting a stack of three slow CPU's that works in parallel and a single fast CPU. Quantitatively we set service rates such that

$$\mu_{fast} = k\mu_{slow}$$

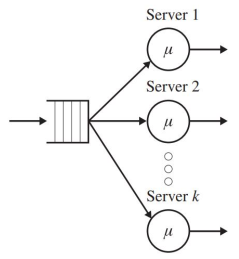


Fig. 3. M/G/k node schematic; slow μ

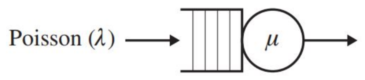


Fig. 4. M/G/k node schematic; fast μ

From queuing theory we have a closed form solution for the average response time for a M/G/1. This is calculated with Laplace Transform and results in the equation in Fig 5

$$\widetilde{T}(s) = \frac{\widetilde{S}(s)(1-\rho)s}{\lambda\widetilde{S}(s) - \lambda + s}.$$

[2],

Fig. 5. M/G/1 overall response time (including queuing time and service time. Note the change of variable:
$s = \lambda - \lambda z$ and $z = 1 - \frac{s}{\lambda}$

Unfortunately, we do not have a closed-form solution for the average response time of M/G/k, and this remains an open long-standing research question. However, we have numerical solutions that consist of analyzing a phase-type distribution (M/PH/k) combined with matrix analysis. Nevertheless, we do not a have model to determine how and which features of the jobs size distribution affect the response time. Lastly, numerical solutions can be unstable (due to singular matrices) if the coefficient of variation [3] of Job size C^2 is high enough. A common approximation is proposed by Lee and Longton [11] in eq 6

From the resulting formula, we can analyze two scenarios Thereafter, according to ρ we can retrieve the best configuration that minimizes response time for a training job of the CNN.

[3]Coefficient of Variation (or relative standard deviation) [10] is the ratio between standard deviation and mean and measures the dispersion of a probability distribution

$$\mathbf{E}\left[T_Q^{\text{M/G/k}}\right] \approx \left(\frac{C^2+1}{2}\right)\mathbf{E}\left[T_Q^{\text{M/M/k}}\right]$$

Fig. 6. M/G/k Waiting time in queue

	high ρ	low ρ
high Variability	M/G/3 is faster than M/G/1	M/G/3 is slower
low Variability	M/G/1 is faster than M/G/3	no difference

TABLE I
EXPECTED OUTCOME

C. Machine Learning: Random Forest Regressor

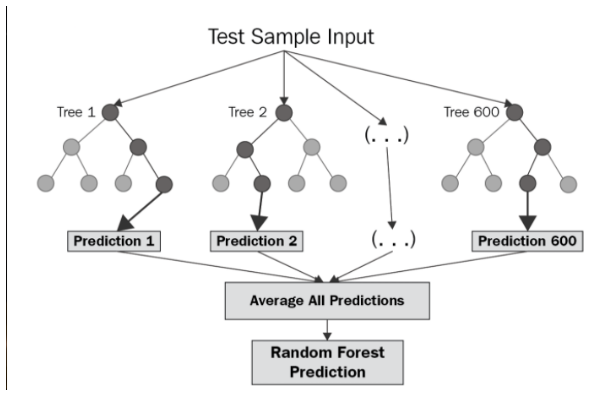


Fig. 7. Random forest regressor

Random Forest is part of the ensemble learning paradigm to provide inference. The core idea is that the combination of weak classifiers or regressors yields better performances than individual ones. According to J. Surowiecki [12] this is due to the diversity of opinion, independence (within a big crowd, each component is not influenced by others), decentralization (enables specialization and local knowledge), aggregation.

In the specific case of Random Forest, each component is a Decision Tree (DT). DTs are testing features sequentially, striving to maximize the information gained at each step. It is based on the definition of Entropy, according to Shannon, the uncertainty in the outcome of an experiment. The greater the degree of uncertainty, the greater the value of the Entropy. So for a uniform distribution, i.e. every single output is equiprobable, the entropy value will be greater. Conversely, the entropy value will be lower for a non-uniform (biased) distribution because some outputs are more likely than others. For example, the Entropy for a fair coin: $p_1 = 0.5$, $p_2 = 0.5$

$$E_1 = \sum_i -p_i log_2 p_i = -0.5 \cdot log_2 \cdot 0.5 - 0.5 \cdot log_2 \cdot 0.5 = 1$$

Instead for a very rigged coin: $p_1 = 0.01$, $p_2 = 0.99$

$$E_2 = \sum_i -p_i log_2 p_i = -0.01 \cdot log_2 \cdot 0.01 - 0.99 \cdot log_2 \cdot 0.99 = 0.08 \tag{1}$$

By choosing the attribute that maximizes the Information Gain, we obtain a lower entropy in the next step. It implies that the probability of the various outcomes in the Subset S_k follows a certain tendency. However, the information gain numerically quantifies how effective it is to ask about any attribute a_i to obtain a more biased distribution of probabilities of the outcomes.

In this research, a Random Forest regressor is trained on the experimental dataset, and it is used to predict the estimated training time for a given configuration.

IV. Experiments

The experiments are executed on the Google Cloud. A pool of 4 nodes was created. Each node had an e2-standard-4 machine with 4 vCPU and 16 GB of memory. We deployed

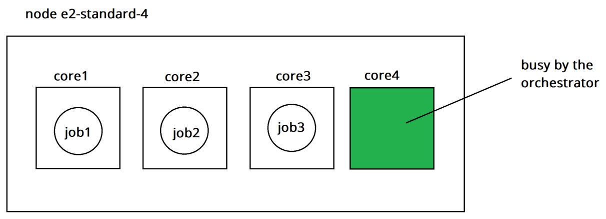
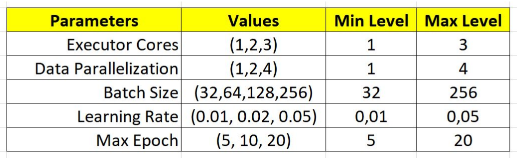


Fig. 8. Google Cloud Kubernetes Setup. Note that one CPU in each node was used to run the kubernetes orchestrator, so was not available to take any training job

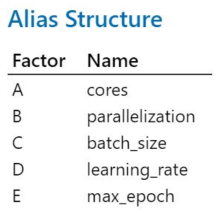
our configuration and jobs to GCloud through Kubernetes and FLTK-testbed [4] Our training dataset is Fashion-MNIST and the network model is Fashion-MNIST-CNN.



Parameters	Values	Min Level	Max Level
Executor Cores	(1,2,3)	1	3
Data Parallelization	(1,2,4)	1	4
Batch Size	(32,64,128,256)	32	256
Learning Rate	(0.01, 0.02, 0.05)	0,01	0,05
Max Epoch	(5, 10, 20)	5	20

Fig. 9. Factors showing all levels

A. Design of Experiments (DoE)



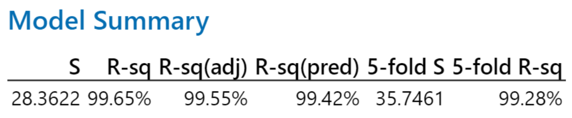
Alias Structure

Factor	Name
A	cores
B	parallelization
C	batch_size
D	learning_rate
E	max_epoch

Fig. 10. Mapping of Factor to Alias

Prior to running the entire 2-factor experiment, we explored the significance of each of the main factors using a 2^{5-2} with 3 replications. It led to running 24 experiments. Resolution for this design is 3 with design Generators D = AB, E= AC. Therefore some main effects are confounded with two-way

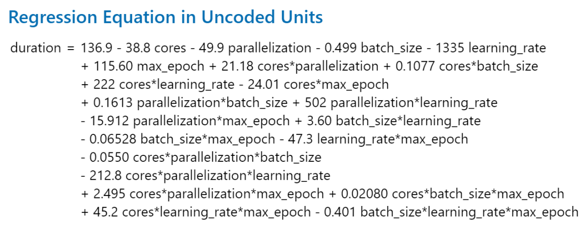
[4]Github Repo url = "https://github.com/JM/Galjaard/fltk-testbed"



Model Summary

S	R-sq	R-sq(adj)	R-sq(pred)	5-fold S	5-fold R-sq
28.3622	99.65%	99.55%	99.42%	35.7461	99.28%

Fig. 11. DoE, R2



Regression Equation in Uncoded Units

```
duration = 136.9 - 38.8 cores - 49.9 parallelization - 0.499 batch_size - 1335 learning_rate
           + 115.60 max_epoch + 21.18 cores*parallelization + 0.1077 cores*batch_size
           + 222 cores*learning_rate - 24.01 cores*max_epoch
           + 0.1613 parallelization*batch_size + 502 parallelization*learning_rate
           - 15.912 parallelization*max_epoch + 3.60 batch_size*learning_rate
           - 0.06528 batch_size*max_epoch - 47.3 learning_rate*max_epoch
           - 0.0550 cores*parallelization*batch_size
           - 212.8 cores*parallelization*learning_rate
           + 2.495 cores*parallelization*max_epoch + 0.02080 cores*batch_size*max_epoch
           + 45.2 cores*learning_rate*max_epoch - 0.401 batch_size*learning_rate*max_epoch
```

Fig. 12. Regression Equation, full factorial design

interactions. From the ANOVA table Fig 19 corresponding to 2^{5-2} factorial design, we observe that all factors have p-value < 0.05. Hence they are all relevant. It led us to run the full 2 factorial experiments. We run the experiments with 3 replications, leading to 96 experiments in total. Again, all main effects are found to be relevant, as proved by the p-values shown in Fig 22 and Fig 20. Mainly the 4 and higher-order interactions were statistically insignificant. Additionally, we have plotted the Pareto chart to have a visual representation of the relative importance of factors and higher-order interactions Fig 13. From the results, the factor 'max_epoch' accounts for the maximum variability in the data, and the factor 'batch_size' has the least variability.

Lastly, to predict the estimated response time, we fit a regression model selecting only the relevant factor and interaction (we excluded 4 and 5 order interactions, according to the aforementioned reasons). We fitted it with 5-fold cross-validation and determined the regression equation Fig 12. Moreover, we proved all the linear regression assumptions by investigating the residuals Fig 14. Lastly, we show with the R2 value that the DoE regression model can explain the variability of the given data in Fig. 11

B. Queuing Theory

We ran simulations on the Google cloud using the two setups - 1 node with 3 cores, 3 nodes with 1 core. The system is modelled as M/G/1 and M/G/k systems with k=3. The arrivals (random variable) are exponentially distributed in both cases (Poisson Process).

The batch size was altered among the jobs to simulate low variability among the jobs, as the batch size was found to account for the lowest variability in the preceding analysis. In the Low variability condition, for both M/G/1 and M/G/3, as utilization increased, they both converged to the same response time. Overall, in the case of low utilization, M/G/1 is seen to be faster than M/G/3, as shown in Fig 16.

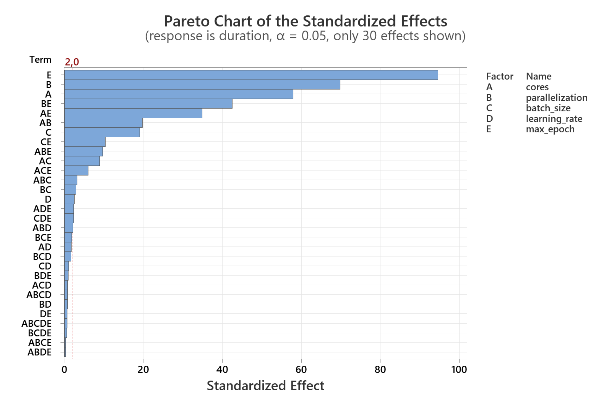


Fig. 13. Pareto chart, 2^5 full factorial design. The red line on the left hand side, sets the threshold of importance of the factors.

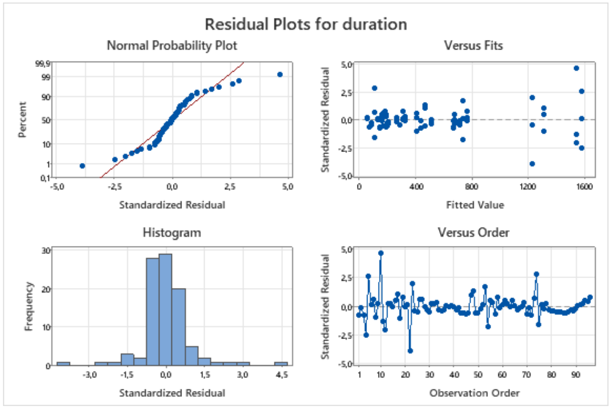


Fig. 14. Residual plots, Full Factorial Design, evaluate whether the linear regression model is good or if a non-linear model could provide a better fit. The histogram shows whether data is skewed and the existence of two outliers. The random pattern for the *Versus Fit* on both sides proves that predictor variables are unrelated to the residuals. Thus all assumptions are satisfied, meaning that a regression model is good.

In the case of a high variability factor, we varied the epoch size in the simulation. It follows the results in the previous section, where epoch was found to be accounting for the maximum variability. For high variability scenarios with low utilization, M/G/1 is better than M/G/3. However, as the utilization increases, the performance of M/G/1 deteriorates significantly, whereas M/G/3 deteriorates slowly before hitting a plateau. It can be seen in Fig 15. Intuitively, for M/G/1, if a big job starts processing, other jobs can not be processed and are stuck in the queue, which drives up the average response time. On the contrary, for M/G/3, even if one big job is held up at one of the servers, other servers are free to process the remaining jobs.

We only showed results for small values of λ. Nevertheless, the trends can be observed. It should be noted that the arrival rate was different in the high and the low variability cases. It was done to ensure that the utilization remains constant as the average time varies between the cases. Thus the Fig 16 and 16 have different values in the x-axis.

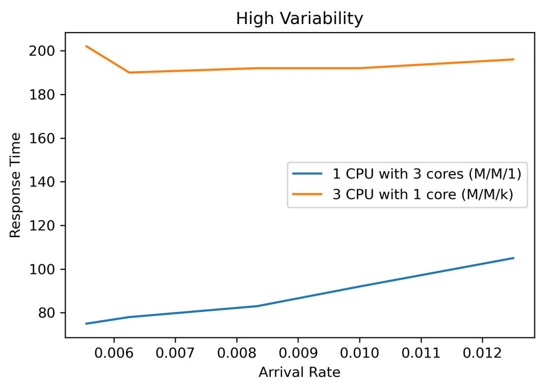


Fig. 15. High variability factor

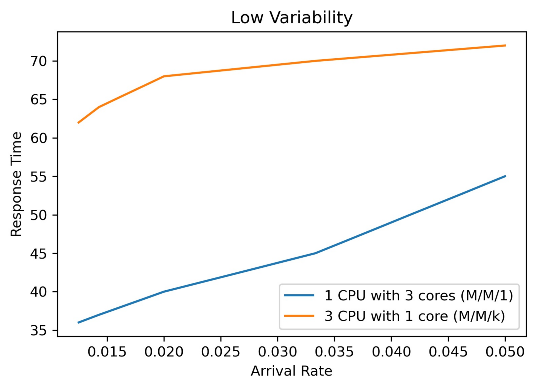


Fig. 16. Low variability factor

C. Machine Learning: Random Forest Regressor

We fit the data with various ML models: Multilayer Perceptron, Linear Regressor, Logistic Regressor, Lasso Regressor and Random Forest. The training data was split into training and test in the ratio of 1:3.

A very recent study [13] shows that R2 [5] can be considered the best metric to determine the goodness of a regression model. By analyzing the R2, we observe that Random Forest is the best model for this task as R2 is the highest for RF.

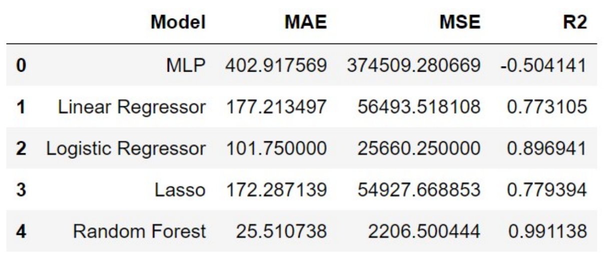
Additionally, for the random forest regressor, we plot the importance of features (according to the information gain principle described before) in fig 18. Finally, the accuracy of the model is reported in the table 17 along with all the other models considered.

Therefore, we will choose Random Forest Regressor for the estimation and optimization in the following section.

V. Optimization Strategy

Our system uses a combination of DoE and Machine Learning models to optimize the training time for the CNN. Our experiments proved that the max epoch is the most significant factor in terms of variability contributions. Therefore we want to optimize that factor while ensuring that training happens within a provided time limit.

[5]R2 is a statistic that will give some information about the goodness of fit of a model. R2 is a statistical measure of how well the regression predictions approximate the real data points in regression. An R2 of 1 indicates that the regression predictions perfectly fit the data. https://en.wikipedia.org/wiki/Coefficient_of_determination .



	Model	MAE	MSE	R2
0	MLP	402.917569	374509.280669	-0.504141
1	Linear Regressor	177.213497	56493.518108	0.773105
2	Logistic Regressor	101.750000	25660.250000	0.896941
3	Lasso	172.287139	54927.668853	0.779394
4	Random Forest	25.510738	2206.500444	0.991138

Fig. 17. Performance metrics

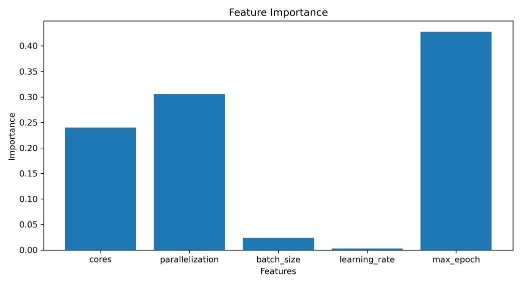


Fig. 18. Feature Importance

To test our system, given a particular time limit, we run two sets of experiments.

- **Optimization Model** In the first set, we use the optimal value given by our proposed algorithm. To calculate the optimal value, we consider as input a lower bound, an upper bound, and the step size for the max epoch, such that we generate a linear space. Then we iterate over all the values in the linear space and predict the training time for each value using an RF and the regression function generated by DoE. Both predictions have equal contributions to the final output. Lastly, the maximum epoch with a predicted time less than the provided time limit is returned to the user.
- **Baseline Model** For the baseline, we keep the other parameters (system and hyperparameters other than max epoch) fixed as in the previous scenario. Instead of getting an optimized value for the max epoch, we choose a random value within the linear space generated by the lower and upper bound. The random value is drawn from a uniform probability density function.

Comparison. As our goal is to satisfy the time limit requirement provided as input, to compare the performances between our optimization algorithm and the baseline, we consider the following **metric**: *We check the percentage of experiments where the response time is less than the provided limit*.

The percentage of training jobs completed within the predetermined time frame in the case of our optimizer is 83%, whereas, for the baseline, it is 50%. Thus we have measured an improvement of 33%.

VI. Conclusions

This work analyzed the relevance of a specific set of features for training a CNN over distributed systems. We show that the most important parameter is the max epoch. We have designed three different models (DoE, Queuing Theory, Random Forest Regressor) to describe the relationship between training time and selected features, to determine the best system and hyperparameter settings. Additionally, we have designed an optimizer powered by our analysis in this research that finds the best settings that satisfy the time limit requirements given as input.

A. Improvements and Future work

- Consider additional factors, e.g. the number of hidden layers.
- Extend this for more detests and different deep learning architectures, e.g. Transformers.
- Include Accuracy in the modelling.
- Include monetary costs to determine the best trade-off between training time setup and monetary costs.

References

[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, pp. 5998–6008, 2017.

[2] N. Kalchbrenner, E. Grefenstette, and P. Blunsom, "A convolutional neural network for modelling sentences," *arXiv preprint arXiv:1404.2188*, 2014.

[3] J. Keuper and F.-J. Preundt, "Distributed training of deep neural networks: Theoretical and practical limits of parallel scalability," in *2016 2nd Workshop on Machine Learning in HPC Environments (MLHPC)*, pp. 19–26, IEEE, 2016.

[4] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang, *et al.*, "Large scale distributed deep networks," *Advances in neural information processing systems*, vol. 25, pp. 1223–1231, 2012.

[5] Y. Li, J. Park, M. Alian, Y. Yuan, Z. Qu, P. Pan, R. Wang, A. Schwing, H. Esmaeilzadeh, and N. S. Kim, "A network-centric hardware/algorithm co-design to accelerate distributed training of deep neural networks," pp. 175–188, IEEE, 2018.

[6] G. X. Yu, Y. Gao, P. Golikov, and G. Pekhimenko, "Habitat: A runtime-based computational performance predictor for deep neural network training," in *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, pp. 503–521, USENIX Association, July 2021.

[7] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International conference on machine learning*, pp. 448–456, PMLR, 2015.

[8] Z. Zhang and M. R. Sabuncu, "Generalized cross entropy loss for training deep neural networks with noisy labels," in *32nd Conference on Neural Information Processing Systems (NeurIPS)*, 2018.

[9] M. Harchol-Balter, *Performance modeling and design of computer systems: queueing theory in action*. Cambridge University Press, 2013.

[10] C. E. Brown, "Coefficient of variation," in *Applied multivariate statistics in geohydrology and related sciences*, pp. 155–157, Springer, 1998.

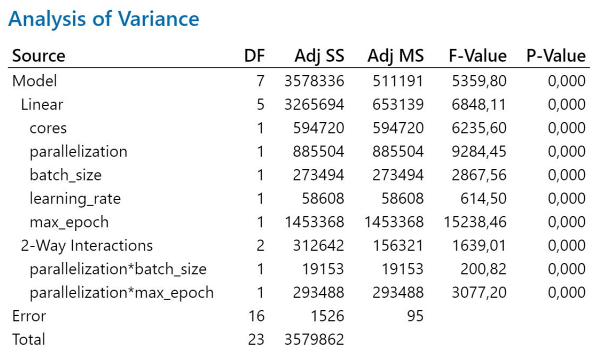
[11] A. Lee and P. Longton, "Queueing processes associated with airline passenger check-in," *Journal of the Operational Research Society*, vol. 10, no. 1, pp. 56–71, 1959.

[12] O. Arazy, W. Morgan, and R. Patterson, "Wisdom of the crowds: Decentralized knowledge construction in wikipedia," in *16th Annual Workshop on Information Technologies & Systems (WITS) Paper*, 2006.

[13] D. Chicco, M. J. Warrens, and G. Jurman, "The coefficient of determination r-squared is more informative than smape, mae, mape, mse and rmse in regression analysis evaluation," *PeerJ Computer Science*, vol. 7, p. e623, 2021.

VII. Appendix

A. 2 level Fractional factorial design

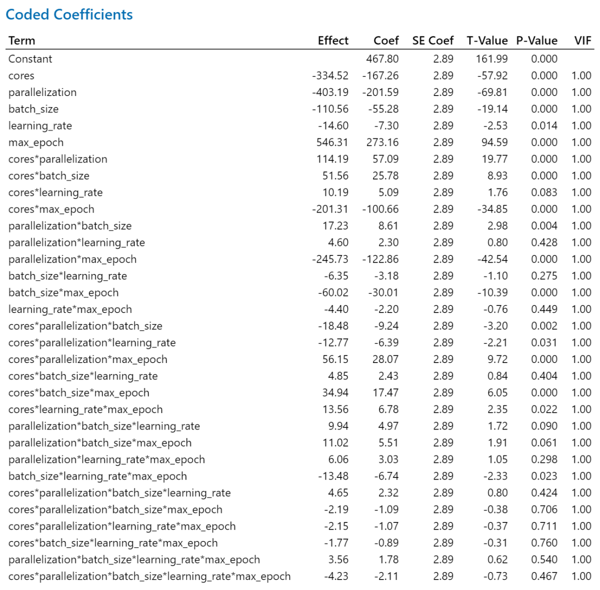


Analysis of Variance

Source	DF	Adj SS	Adj MS	F-Value	P-Value
Model	7	3578336	511191	5359,80	0,000
Linear	5	3265694	653139	6848,11	0,000
cores	1	594720	594720	6235,60	0,000
parallelization	1	885504	885504	9284,45	0,000
batch_size	1	273494	273494	2867,56	0,000
learning_rate	1	58608	58608	614,50	0,000
max_epoch	1	1453368	1453368	15238,46	0,000
2-Way Interactions	2	312642	156321	1639,01	0,000
parallelization*batch_size	1	19153	19153	200,82	0,000
parallelization*max_epoch	1	293488	293488	3077,20	0,000
Error	16	1526	95		
Total	23	3579862			

Fig. 19. ANOVA, 2^{5-2} fractional factorial design, run with 3 replications

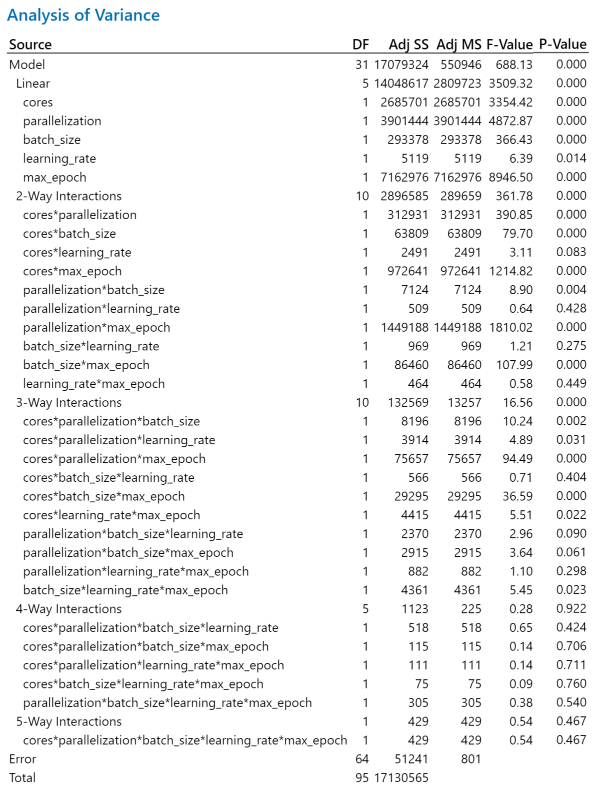
B. 2 level full factorial design



Coded Coefficients

Term	Effect	Coef	SE Coef	T-Value	P-Value	VIF
Constant		467.80	2.89	161.99	0.000	
cores	-334.52	-167.26	2.89	-57.92	0.000	1.00
parallelization	-403.19	-201.59	2.89	-69.81	0.000	1.00
batch_size	-110.56	-55.28	2.89	-19.14	0.000	1.00
learning_rate	-14.60	-7.30	2.89	-2.53	0.014	1.00
max_epoch	546.31	273.16	2.89	94.59	0.000	1.00
cores*parallelization	114.19	57.09	2.89	19.77	0.000	1.00
cores*batch_size	51.56	25.78	2.89	8.93	0.000	1.00
cores*learning_rate	10.19	5.09	2.89	1.76	0.083	1.00
cores*max_epoch	-201.31	-100.66	2.89	-34.85	0.000	1.00
parallelization*batch_size	17.23	8.61	2.89	2.98	0.004	1.00
parallelization*learning_rate	4.60	2.30	2.89	0.80	0.428	1.00
parallelization*max_epoch	-245.73	-122.86	2.89	-42.54	0.000	1.00
batch_size*learning_rate	-6.35	-3.18	2.89	-1.10	0.275	1.00
batch_size*max_epoch	-60.02	-30.01	2.89	-10.39	0.000	1.00
learning_rate*max_epoch	-4.40	-2.20	2.89	-0.76	0.449	1.00
cores*parallelization*batch_size	-18.48	-9.24	2.89	-3.20	0.002	1.00
cores*parallelization*learning_rate	-12.77	-6.39	2.89	-2.21	0.031	1.00
cores*parallelization*max_epoch	56.15	28.07	2.89	9.72	0.000	1.00
cores*batch_size*learning_rate	4.85	2.43	2.89	0.84	0.404	1.00
cores*batch_size*max_epoch	34.94	17.47	2.89	6.05	0.000	1.00
cores*learning_rate*max_epoch	13.56	6.78	2.89	2.35	0.022	1.00
parallelization*batch_size*learning_rate	9.94	4.97	2.89	1.72	0.090	1.00
parallelization*batch_size*max_epoch	11.02	5.51	2.89	1.91	0.061	1.00
parallelization*learning_rate*max_epoch	6.06	3.03	2.89	1.05	0.298	1.00
batch_size*learning_rate*max_epoch	-13.48	-6.74	2.89	-2.33	0.023	1.00
cores*parallelization*batch_size*learning_rate	4.65	2.32	2.89	0.80	0.424	1.00
cores*parallelization*batch_size*max_epoch	-2.19	-1.09	2.89	-0.38	0.706	1.00
cores*parallelization*learning_rate*max_epoch	-2.15	-1.07	2.89	-0.37	0.711	1.00
cores*batch_size*learning_rate*max_epoch	-1.77	-0.89	2.89	-0.31	0.760	1.00
parallelization*batch_size*learning_rate*max_epoch	3.56	1.78	2.89	0.62	0.540	1.00
cores*parallelization*batch_size*learning_rate*max_epoch	-4.23	-2.11	2.89	-0.73	0.467	1.00

Fig. 20. Effects Table, 2^5 full factorial design



Analysis of Variance

Source	DF	Adj SS	Adj MS	F-Value	P-Value
Model	31	17079324	550946	688.13	0.000
Linear	5	14048617	2809723	3509.32	0.000
cores	1	2685701	2685701	3354.42	0.000
parallelization	1	3901444	3901444	4872.87	0.000
batch_size	1	293378	293378	366.43	0.000
learning_rate	1	5119	5119	6.39	0.014
max_epoch	1	7162976	7162976	8946.50	0.000
2-Way Interactions	10	2896585	289659	361.78	0.000
cores*parallelization	1	312931	312931	390.85	0.000
cores*batch_size	1	63809	63809	79.70	0.000
cores*learning_rate	1	2491	2491	3.11	0.083
cores*max_epoch	1	972641	972641	1214.82	0.000
parallelization*batch_size	1	7124	7124	8.90	0.004
parallelization*learning_rate	1	509	509	0.64	0.428
parallelization*max_epoch	1	1449188	1449188	1810.02	0.000
batch_size*learning_rate	1	969	969	1.21	0.275
batch_size*max_epoch	1	86460	86460	107.99	0.000
learning_rate*max_epoch	1	464	464	0.58	0.449
3-Way Interactions	10	132569	13257	16.56	0.000
cores*parallelization*batch_size	1	8196	8196	10.24	0.002
cores*parallelization*learning_rate	1	3914	3914	4.89	0.031
cores*parallelization*max_epoch	1	75657	75657	94.49	0.000
cores*batch_size*learning_rate	1	566	566	0.71	0.404
cores*batch_size*max_epoch	1	29295	29295	36.59	0.000
cores*learning_rate*max_epoch	1	4415	4415	5.51	0.022
parallelization*batch_size*learning_rate	1	2370	2370	2.96	0.090
parallelization*batch_size*max_epoch	1	2915	2915	3.64	0.061
parallelization*learning_rate*max_epoch	1	882	882	1.10	0.298
batch_size*learning_rate*max_epoch	1	4361	4361	5.45	0.023
4-Way Interactions	5	1123	225	0.28	0.922
cores*parallelization*batch_size*learning_rate	1	518	518	0.65	0.424
cores*parallelization*batch_size*max_epoch	1	115	115	0.14	0.706
cores*parallelization*learning_rate*max_epoch	1	111	111	0.14	0.711
cores*batch_size*learning_rate*max_epoch	1	75	75	0.09	0.760
parallelization*batch_size*learning_rate*max_epoch	1	305	305	0.38	0.540
5-Way Interactions	1	429	429	0.54	0.467
cores*parallelization*batch_size*learning_rate*max_epoch	1	429	429	0.54	0.467
Error	64	51241	801		
Total	95	17130565			

Fig. 21. ANOVA, 2^5 full factorial design

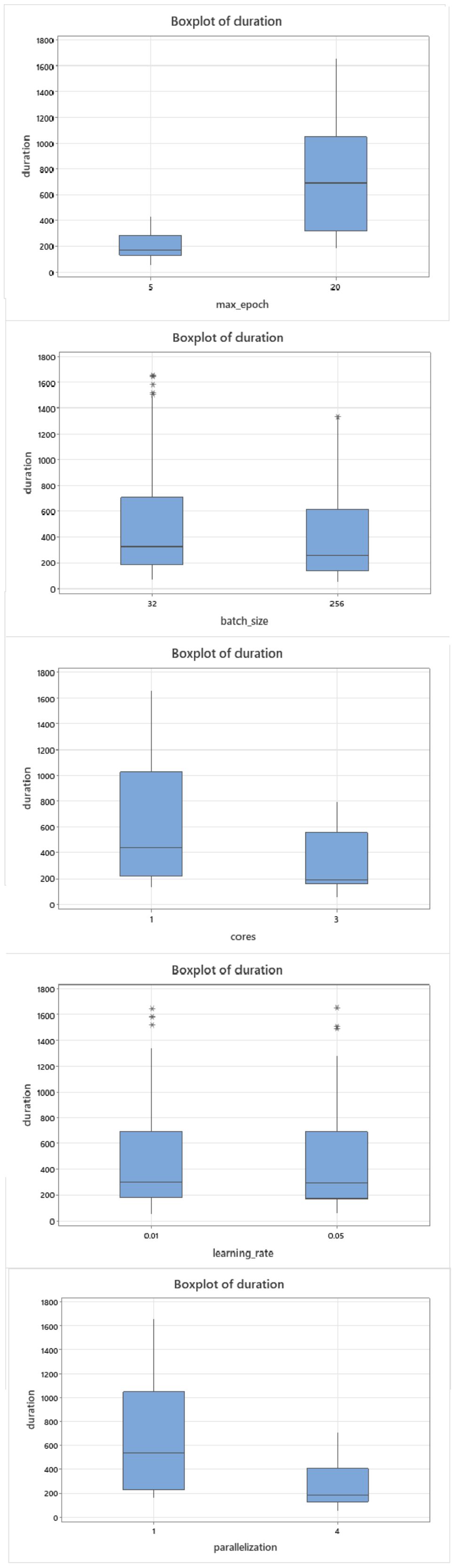


Fig. 22. Distribution of response time with respect to the 5 factors. Each boxplot shows the median, range, interquartile, and outliers