Big Data preprocessing and ML training Report

Bernardo Saab Martiniano de Azevedo Bernardo.Azevedo.2@city.ac.uk

Colab Notebook link: https://colab.research.google.com/drive/14XvoYrPjdUBtdcqrHdJDH47Odlyw68EO?usp=sharing

1b)

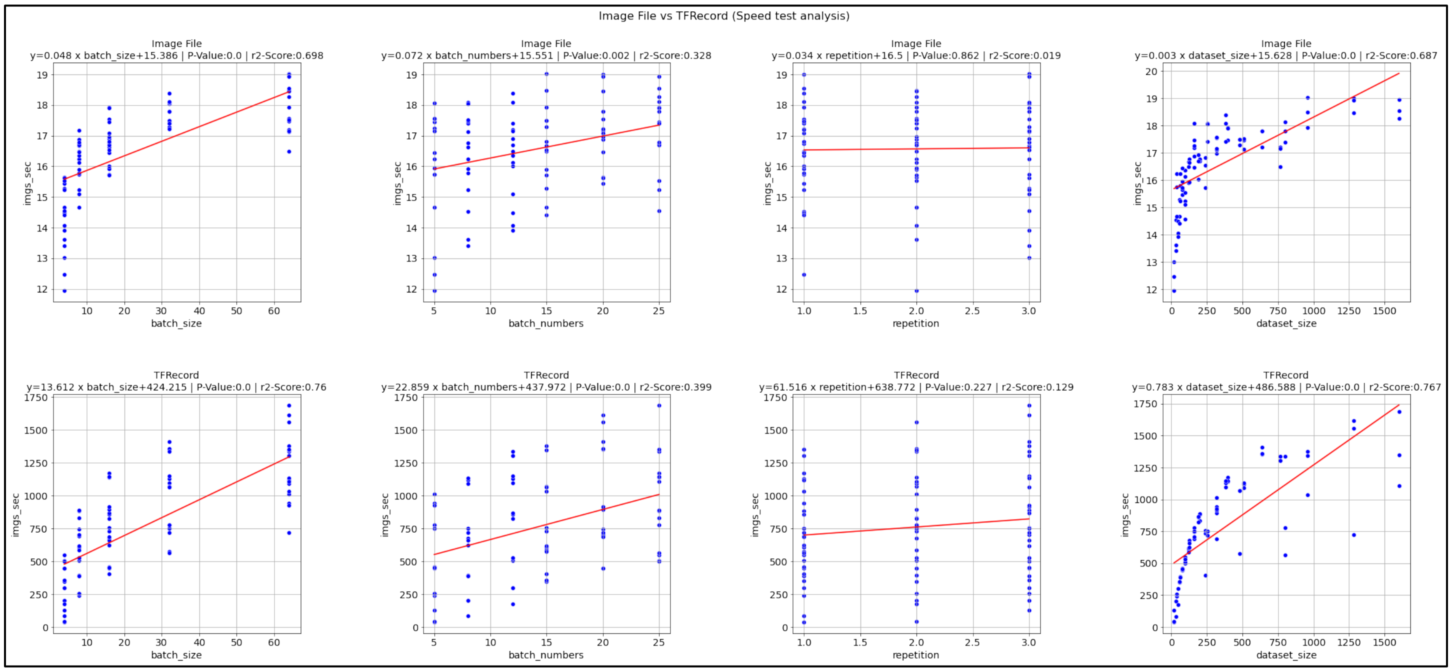


Figure 1: Images per second from Image and TFRecord files as a function of batch and dataset parameters.

2b)

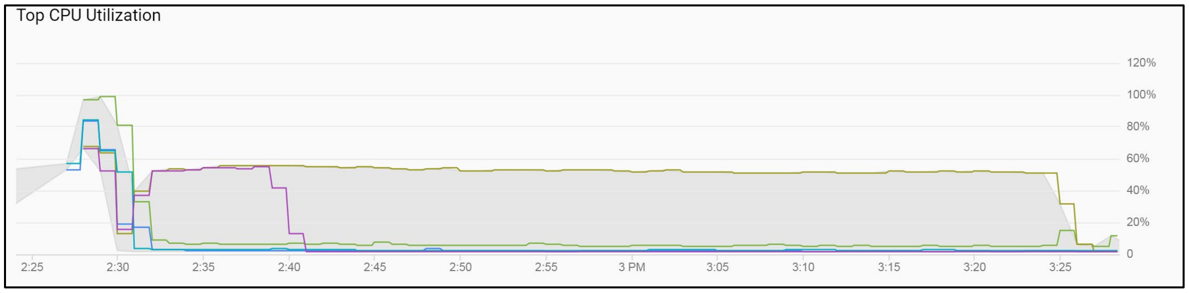


Figure 2: CPU Utilization.

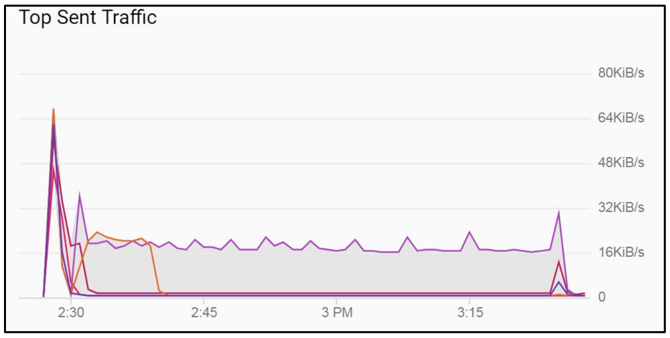


Figure 3: Sent Traffic (Network).

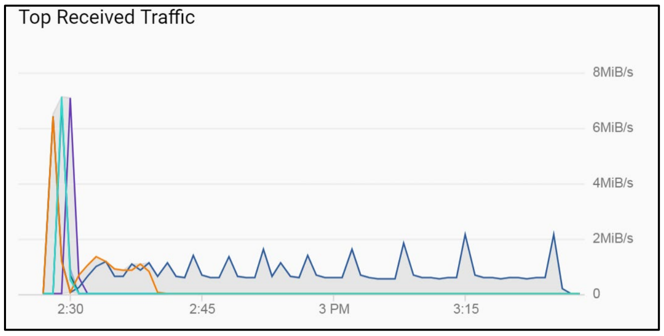


Figure 4: Received Traffic (Network).

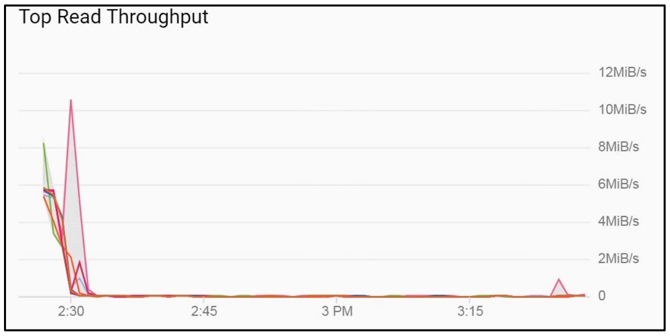


Figure 5: Read Throughput (Disk).

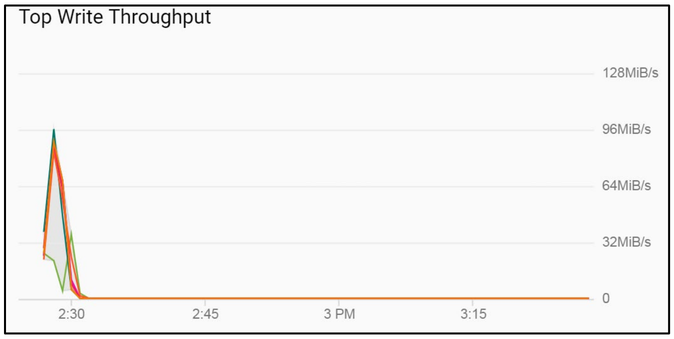


Figure 6: Write Throughput (Disk).

In Figure 2 all the nodes were executing due to the cluster initialization with a single worker node predominantly contributing to CPU. As expected, both read and write throughputs (Figures 5/6) have low levels because we are accessing data from the bucket and using the disk, except from a small utilization in the end as likely no cache was used. The same behavior is seen in Figures 3/4 when network traffic peaks at the beginning, therefore only 2 nodes had network communication for most part of the job. This indicates a default partition was utilized (2) for workers to communicate across the network, suggesting an optimization or scale down application to eliminate idle workers and save costs.

2c)

The effects of caching on cluster cost and time efficiency when tasks are executed repeatedly are tremendous and the literature contemplates various advantages by using memory to persist across different intermediate computations of RDDs, such as aggregations and mappings. Spark uses cache to persist each partition entirely, this means if portions of partitions are meant to be cached the whole partition should be evaluated and persisted. Therefore, each partition can be utilized again by the nodes in different actions (e.g. collect).

Figure 7 shows results from the same job with 4 different configurations. The first has no caching and parallelization, the second only caching, the third parallelization and both caching and parallelization are used at last. While the first 2 jobs are similar in total execution time due to the default parallelization, in jobs 3 and 4 we note several worker nodes utilizing the CPU at the same time and even with network peaks the overall traffic is negligible for sending data between executors the execution time. The total time was reduced and aggregations or grouping operations optimized, mostly because partitions were defined as 2 times the total cores, as per suggestion that the number of partitions should be ideally 2- to 4-fold [1] of the cores.

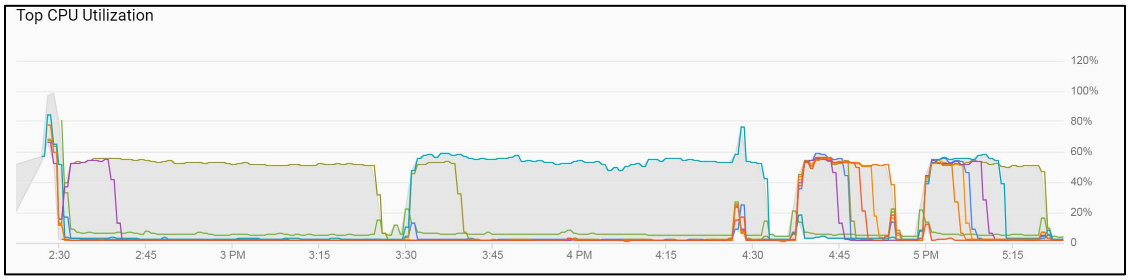


Figure 7: CPU Utilization.

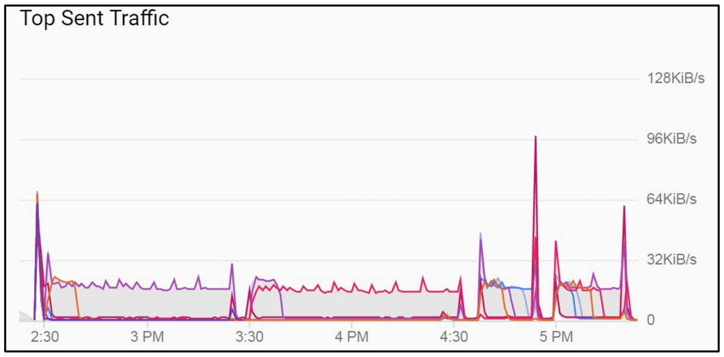


Figure 8: Sent Traffic (Network).

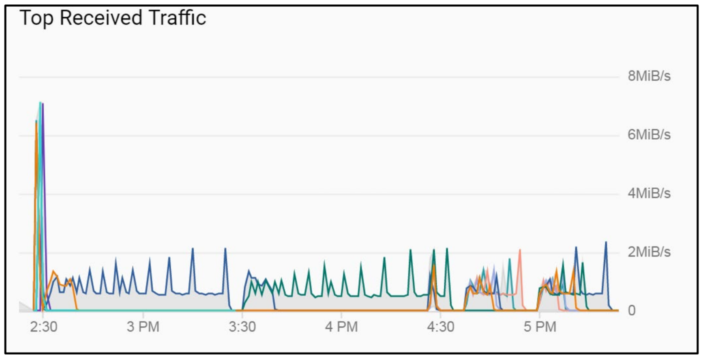


Figure 9: Received Traffic (Network).

Lastly, we recommend the wise use of cache and partitions, thus we reinforce the better use of cluster resources in important tasks.

2d)

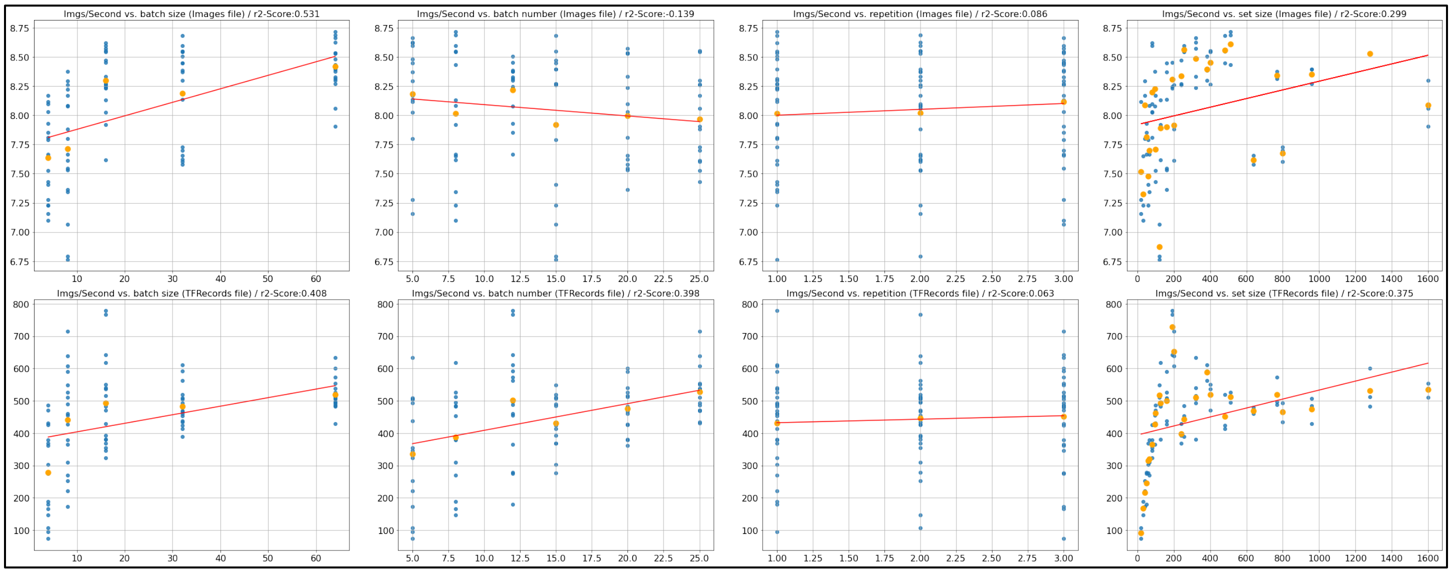


Figure 10: Image and TFRecord files throughput and average throughput (orange) as a function of batch and dataset parameters (Cloud) – Parallelisation and Caching enabled.

The same experiment as in task 1 was performed in a cluster with 8vCPUs (Figure 10) with caching and 16 partitions. We observed better results from increasing batch size in the cluster contrast to a single node, mostly due to partitioning. There was a degrading performance of batches of images throughput as opposed to TFRecords, which are optimised for multiprocessing due to its dense matrix representation.

As informed in GCS documentation [2] we must reason for multitenancy and IO limitations to perform read/write operations in buckets, which have initial capacity for requests/second but can be distributed across machines. Moreover, IOPS and throughput influence in disk performance, the first is dictated by the sequential or random-access pattern and throughput is limited by disk types, usually higher for SSD over persistent disks. For regional disks there is a maximum IOPS of 0.75/1.5 and 30/30 and throughput(MB/s) of 0.12/0.12 and 0.48/0.48 per GB for read/write persistent and SSDs operations respectively, meaning we could explore disks usage for certain tasks given a theoretical sequential read of 1s/GB and 20s/GB for SSD and persistent disks [3]. If cluster resources are not restricting the dataset size and operations, we can scale and use ML algorithms to extract higher batches from data to update parameters and improve results, possibly using CPU caching when available. If a dataset becomes larger than available machines in one region, we can distribute workload over other regions machines considering network is not a limiting factor.

The throughput for dataset_size has an asymptotic curve and is plateaued for image files with less variation after initial figures, so we estimate that a larger dataset_size yields better performance for preprocessed TFRecord files and follows a more linear relationship. The effect of increasing repetitions is not meaningful for both types and is reflected in low R2 scores indicating that no caching mechanism was utilized. The linear model is not an adequate representation given the low coefficient of correlation. In addition, we could explore other functions such as higher order polynomials to find a better approximation.

3c)

i) In labs we used Spark on Google Colab similarly to a local instance with a shared working memory and RDD pipelines were developed in small and optimised steps for a single node. Therefore, operations such as filter and reduceByKey transformations were frequently used, and complexity was further reduced with hashing and smaller datasets. We also used RDD distributed through multiple cores, but limited to one machine, consequently we could not benefit from synchronization of multiple workers and RDDs partitions. However, in task 3 we performed experiments with Cloud Dataproc, which uses VM instances and allow a variety of cluster configurations based on CPUs, machine types, disk sizes within multiple nodes and networking selections. By running Spark in clusters, we utilized network to improve traffic and reduce communication between machines and adapt clusters to the application needs. Other advantages from

Dataproc can be summarized by its simplified deployment and capabilities of monitoring, connectivity and scaling according to tasks.

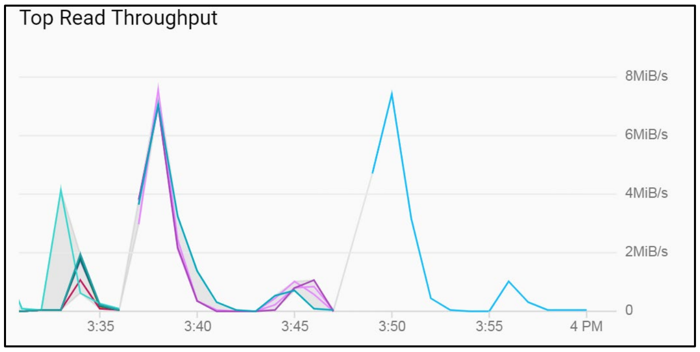


Figure 11: Read Throughput (Disk).

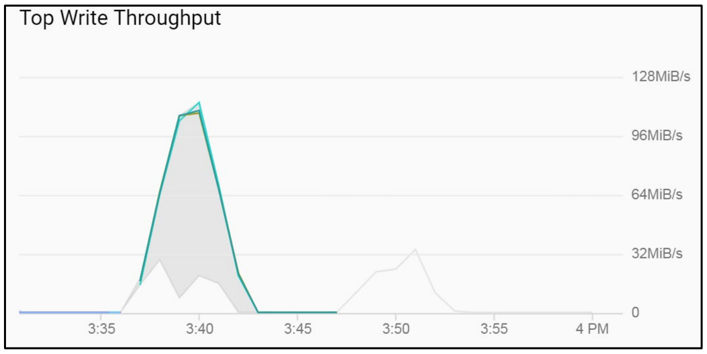


Figure 12: Write Throughput (Disk).

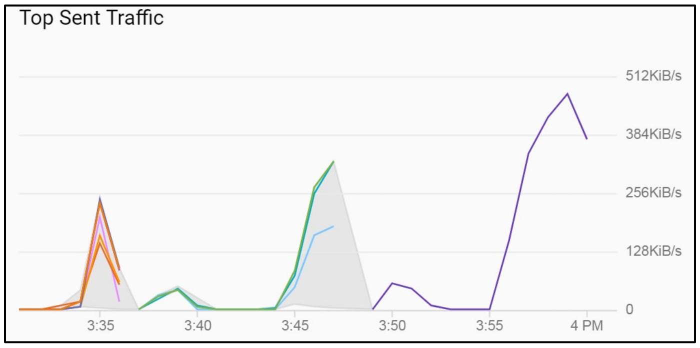


Figure 13: Sent Traffic (Network).

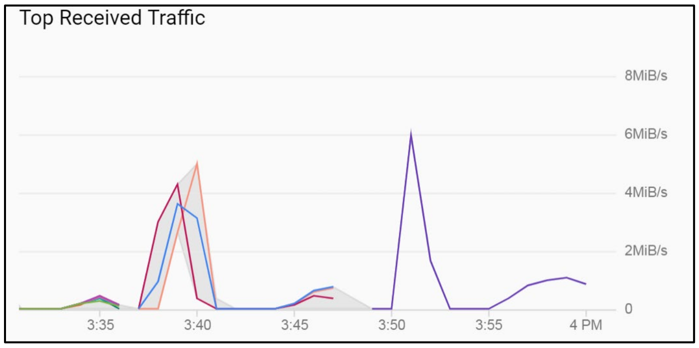
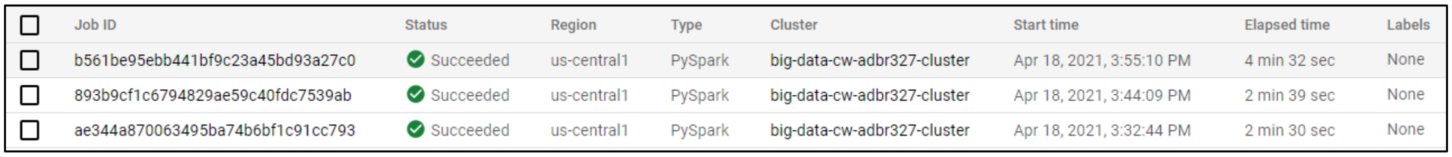


Figure 14: Received Traffic (Network).



☐	Job ID	Status	Region	Type	Cluster	Start time	Elapsed time	Labels
☐	b561be95ebb441bf9c23a45bd93a27c0	✓ Succeeded	us-central1	PySpark	big-data-cw-adbr327-cluster	Apr 18, 2021, 3:55:10 PM	4 min 32 sec	None
☐	893b9cf1c6794829ae59c40fdc7539ab	✓ Succeeded	us-central1	PySpark	big-data-cw-adbr327-cluster	Apr 18, 2021, 3:44:09 PM	2 min 39 sec	None
☐	ae344a870063495ba74b6bf1c91cc793	✓ Succeeded	us-central1	PySpark	big-data-cw-adbr327-cluster	Apr 18, 2021, 3:32:44 PM	2 min 30 sec	None

Table 1: Jobs execution time.

ii) Considering the baseline cluster we ran 2 other experiments. The first had 3-workers and 1-master with double vCPUs each and twice of disk space for workers, and the second a single machine with 4-vCPUs in master node. In Table 1 we note running time was practically equal with small increase for the latter in view of job queuing time. Figures 11/14 disk and network measures shows the read throughput peaked during cluster initialization and values over 4-mebibytes can be ignored. It is evident the throughput is optimal for the first experiment when all the workers are combined, meaning that a single node can reach its throughput limit if more data is allowed and vertical scaling is not an option. Therefore, the baseline cluster resources are shared between workers efficiently and data can be divided in smaller sets to be distributed and synchronized after workers calculation, thus the results would be transferred over the network to a single node output.

As we did not vary the number of partitions given that the number of cores was the same for all the experiments one can conclude that for our task a cluster with more nodes outperforms clusters with the same number of cores but distributed in fewer machines. This is a result of the distributed programming and in-memory Spark nature as well as network balance across dedicated machines. However, vertical scaling can also be an option when data do not fit in memory.

4c)

Strategy	Cluster Configuration	Batch size	Epochs	Wall clock training time (Seconds)	Validation Accuracy (%)	Validation Loss
Not distributed	9-node / 9 K80	32	60	734.296124	19.49%	1.669247
Not distributed	9-node / 9 K80	64	60	376.309376	16.25%	1.724697
Not distributed	9-node / 9 K80	128	60	196.501019	18.44%	1.674609
Not distributed	9-node / 9 K80	256	60	100.288697	19.53%	1.648392
One Device	1 single node / 1 K80	32	60	397.939221	26.79%	1.617378
One Device	1 single node / 1 K80	64	60	357.614018	28.13%	1.618965
One Device	1 single node / 1 K80	128	60	369.495762	45.16%	1.610645
One Device	1 single node / 1 K80	256	60	342.716976	45.31%	1.31029
Mirrored	1 single node / 8 K80	64	60	254.529133	39.22%	1.549793
Mirrored	1 single node / 8 K80	128	60	199.997756	35.31%	1.567807
Mirrored	1 single node / 8 K80	256	60	163.447936	53.91%	1.309216
Mirrored	1 single node / 8 K80	512	60	151.499459	59.18%	1.355762
Multi-worker Mirrored	10-node / 10 K80	64	60	428.351528	27.66%	1.614892
Multi-worker Mirrored	10-node / 10 K80	128	60	276.197418	29.22%	1.46155
Multi-worker Mirrored	10-node / 10 K80	256	60	154.457371	48.24%	1.52962
Multi-worker Mirrored	10-node / 10 K80	512	60	97.072375	33.79%	1.60362

Table 2: Baseline strategy x Distributed Strategies.

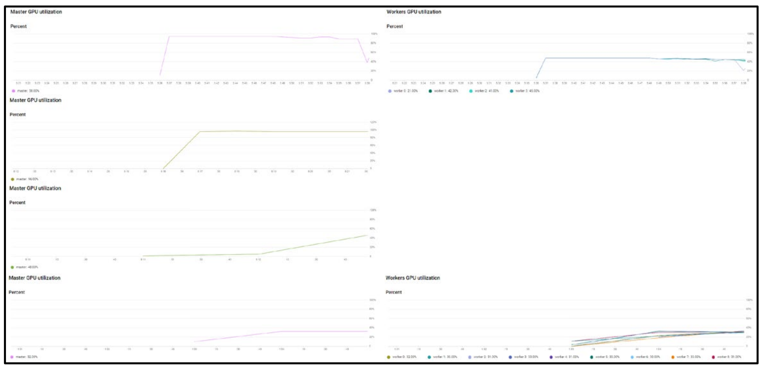


Figure 15: GPUs Consumption for Batch size:256 (None/One Device/ Mirrored /MW-Mirrored), proportionally scaled in x-axis.

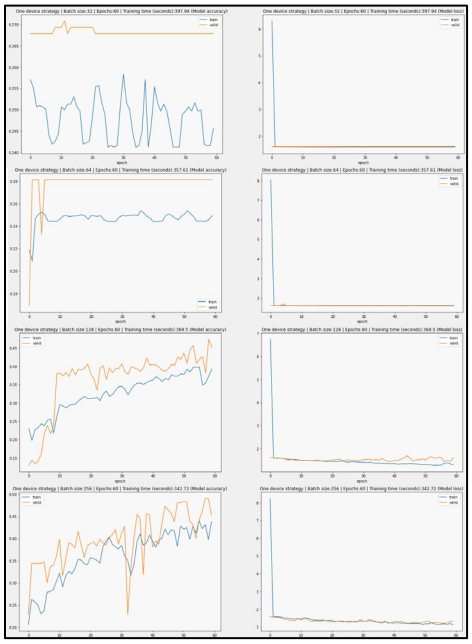


Figure 16: One Device Strategy 60 epochs epochs (Batch size: 64, 128, 256, 512).

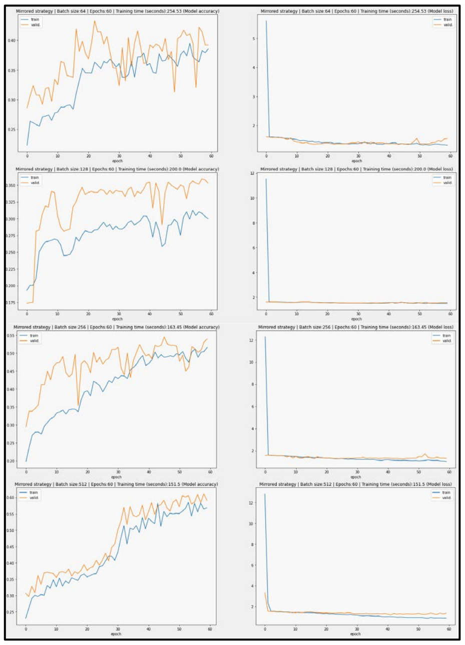


Figure 17: Mirrored Strategy 60 epochs (Batch size: 64,128,256,512).

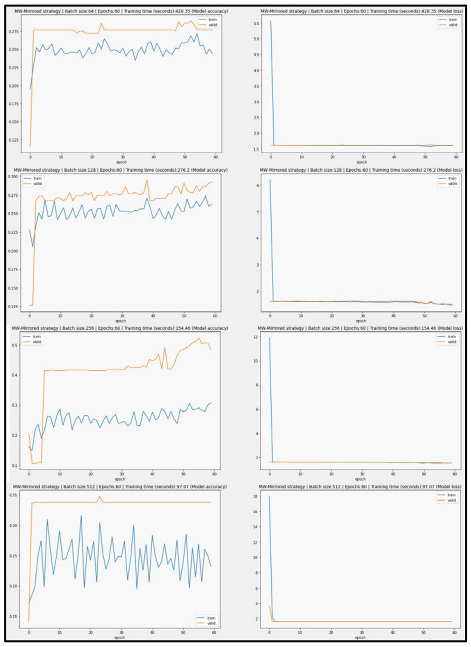


Figure 18: Multi-worker MirroredStrategy 60 (Batch size:32,64,128,256).

Training networks in single machines and GPU can be adequate for some tasks, but others can be impacted because deep learning benefits from larger datasets and networks. We observed in task 4a that higher batch sizes and epochs result in longer training times if training is not distributed. Therefore, we explored distributed strategies to scale out by using more machines and GPUs. The OneDeviceStrategy distributes variables across a unique device and is mostly used for code validation. MirroredStrategy accepts multiple GPUs, however only one machine is allowed, and copy of variables are created and updated on each GPU. The MultiWorkerMirroredStrategy is comparable to MirroredStrategy, but several GPUs can be assigned to multiple machines. Thus, training data can be split across replicas, so we expect faster training from model weights updates comparing to baseline, allowing us to scale batch size as opposed to decrease in learning rate [4].

Except from OneDevice that considered 32/256 batches, we utilized higher batches (64/512) for the others. We maintained the global batch size constant for experiments provided the dataset size and increased validation_split to 0.20 to afford more training data. Mirrored and Multi-worker batches are split among workers in local-batches and each GPU device executes a copy of the model. We tested with 5 epochs and further increased to 60, though empirically we believe higher epochs would produce better accuracy.

From model history metrics, we note that generally smaller batches produce poor accuracy, because training data might be unrepresentative given the limited examples compared to validation. Moreover, training accuracy fluctuates meaning that training is not stabilized, thus, to improve learning speed one should increase the batch size or adjust the learning rate. Overfit was seen for higher batches after 30 iterations, although not severely given the small gap between both curves. The best accuracy and time results were seen for Multi-worker and Mirrored with batch size of 256, and high score of 59% and 151s for the latter, respectively. For nearly all the curves the validation accuracy was better than training indicating that validation set might be easier for prediction than training. If more training time was available, we could cross-validate the model or use different splits.

GPUs utilization (Figure 10) for distributed strategies showed that OneDevice had virtually 100%, Mirrored was better between 40% and 60% and Multi-worker achieved impressive results with close to 30% for most time. Comparing utilization with accuracy and training times, we notably see the advantages of distributed strategies and GPU ability to process multiple computations for higher batch sizes, leading to faster convergence and better performance.

5a)

In tasks 2/3 we detected gains from cloud jobs execution, therefore we hypothesize a better performance would be obtained with CherryPick adaptation [5], in contrast to other methods (e.g. random search). Considering both tasks benefited from distributing programming and running times diverged with various cloud configurations, we believe BO applies to select optimal cloud configurations and reduce uncertainty from inputs. Moreover, authors proved that worst case configurations can cost at most 12-times more than the minimum choice. GCP offers machine recommendations to some extent, but it relies on monitoring metrics for the previous 8 days only. Given the unknown objective function and non-deterministic behavior of clouds, we consider it is expensive to run experiments over all possible and constantly changing configurations. VM instances cost are written as function of task and time, therefore we suggest a running time constraint and discretization of candidates to minimize the search. Even having O(N^4) complexity, less choices are necessary to converge given expected results from confidence intervals. Some candidates cost more than similar ones with distinct architecture [6], consequently to minimize cloud usage and find diminishing return regions we estimate the time/cost for running configurations, search time/cost for sampled choices from all configurations and cluster utilization.

Given the relationship between running time against cloud resources, we observed a non-linear behavior when resources were increased in different proportion over time. Hence, CherryPick would interpret this and estimate noise in GCP, which occurs even when resources are in single regions. Therefore, we could execute small workloads on numerous configurations to find the high percentile.

If tasks are not executed frequently the approach offers ineffective results from search cost, but tasks can be scheduled on regular basis or expected improvement can be adjusted given the periodicity. Task 2 allows some inaccuracy from near-optimal because of repetitions provided similar inputs, but task 3 might need to be recomputed since change in files structure can make the workloads less representative. In task 3 we could explore configurations closer to the initial to improve accuracy, thus more workers in detriment of less workers and faster machines. Effects would be greater in task 2 if applied to deep learning, because of combined techniques of large-batch and Bayesian optimisation.

5b)

There are 3 distinct scenarios for data processing (batch, stream and online). Studies in [4] propose a relation between batch size and training set size in order to find optimal batch size which are also expressed as a function of the fluctuation in training dynamics. They achieved efficient results when maintaining constant learning rate and increasing batch size until approximating 10% of the dataset, and from that point learning rates were decreased. In this work we performed similar approaches with increases in batch size across different strategies until a certain threshold given the dataset size (3670 samples), even without hyperparameter tuning. However, for a better understanding of the network and to prevent overfitting we should have considered an early stop criteria and dropout variation. We should be careful as accuracy can become worse when batch sizes are increased, thus an optimal batch size can be found according to proportions of the learning rate, which in task 4 was given by Adam algorithm, that computes individual adaptive learning rates (defaults to 0.001) for parameters. We compare our experiments of increasing batch size schedules to learning rate schedules decrease, but with less parameter updates for the former. Starting from minibatches we increased batch size and approximated to batch processing, consequently faster training was achieved as more GPUs were included to parallelise workload. In addition, we could also verify if adjustments to learning rate were necessary.

Despite the differences between scenarios it is debatable where the border separates each. In the same paper [4] authors said that small batches can generalize better for test sets, however as we used Adam instead of SGD this was not perceived. Studies [7] discuss advantages from smaller batches as for example a better generalization and less memory consumption. Undoubtedly, if a batch size of 1 is used we expect extreme noise as it is based in a unique random data point and when batch size has all training data it can be harmful for the generalisation performance, however the training regime can be adapted by changes in batch normalization, learning rate and number of iterations [8].

Despite not experimenting small enough batches to consider as online batching in task 4, we noted that smaller batches produced lower accuracy and high losses at the cost of higher running times. In summary, we confirmed our experiments were aligned to theory because the parameter updates given by training times were reduced and accuracy improved when scaling the batch size, furthermore the same benefits from decaying the learning rate were achieved.

REFERENCES

[1] “Spark Programming Guide - Spark 2.1.1 Documentation.” https://spark.apache.org/docs/2.1.1/programming-guide.html (accessed Apr. 16, 2021).

[2] “Request rate and access distribution guidelines | Cloud Storage,” *Google Cloud*. https://cloud.google.com/storage/docs/request-rate (accessed Apr. 18, 2021).

[3] 262588213843476, “Latency numbers every programmer should know,” *Gist*. https://gist.github.com/hellerbarde/2843375 (accessed Apr. 24, 2021).

[4] S. L. Smith, P.-J. Kindermans, C. Ying, and Q. V. Le, “DON’T DECAY THE LEARNING RATE, INCREASE THE BATCH SIZE,” p. 11, 2018.

[5] O. Alipourfard and M. Yu, “CherryPick: Adaptively Unearthing the Best Cloud Configurations for Big Data Analytics,” p. 15.

[6] “Pricing | AI Platform Training,” *Google Cloud*. https://cloud.google.com/ai-platform/training/pricing (accessed Apr. 24, 2021).

[7] D. Masters and C. Luschi, “Revisiting Small Batch Training for Deep Neural Networks,” *ArXiv180407612 Cs Stat*, Apr. 2018, Accessed: Apr. 23, 2021. [Online]. Available: http://arxiv.org/abs/1804.07612.

[8] “Hoffer et al. - 2018 - Train longer, generalize better closing the gener.pdf.” .