Project 3: DNN Hyperparameter Optimization via NNI

Oliver Fowler and Brian Park

North Carolina State University, Computer Science 591/791-025

October 2022

1 HPO Methods

We use NNI to do hyperparameter optimization [1]. We tried multiple tuners such as TPE, Evolution, and Hyperband [2, 3, 4]. We use a simple shallow multi-layer perceptron on MNIST Fashion dataset. This was an example provided by PyTorch. We chose this due to time constraints as we had issues with setting up and allocating nodes on ARC cluster. Other classes were also doing compute intensive assignments, making it difficult to train a more complex model. When we had time left over after doing the required portions, we used a more resource demanding DNN, VGG-19 on CIFAR-10 [5].

2 HPO and Hardware Configurations

2.1 HPO Configurations

For the tuner configurations we used TPE, Evolution, and Hyperband [2, 3, 4]. For hyperparamters to tune, we set the `optimize` mode to `maximize` and let the tuner handle the rest. We set parameters for the learning rate and momentum. For the secondary configuration, we also experimented and treated the number of features and batch size as hyperparameters. Varying both the feature size and batch size will give us a sense of speed. Smaller batches will generally lead to slower training times but higher accuracy, while larger batches will be able to exploit parallelism to get faster training times, but lower accuracy.

For VGG-19, we repeated the configurations, but number of features is not implemented since we loaded a premade model from PyTorch.

The configurations is as follows:

- Learning Rate: Log uniformly sampled between $0.0001, 0.1$
- Momentum: Uniformly sampled between $0, 1$
- Features: $128, 256, 512, 1024$
- Batch Size: $1, 4, 32, 64, 128, 256$

2.2 Hardware Configurations

We had difficulties running the hyperparameter tuner experiments successfully on the ARC cluster [6]. We were fortunate enough to get access to the PSC's bridges-2 supercomputer, which uses 8 NVIDIA V100 SXM2 per node, part of a DGX system [7, 8]. The DGX system uses has two Intel(R) Xeon(R) Gold 6148 CPU @ 2.40GHz CPUs. When allocated via the name `GPU-Shared`, we will only connect to 1 CPU and 1 GPU, as only 4 GPUs are connected via PCIe per CPU node. We use a single GPU, as we're not aware if NNI can utilize multiple GPU in a intra-node setting. Oliver used his RTX 3080 Ti and Brian was able to use his M1 Mac to do the hyperparameter search for each configuration and tuner concurrently. Once we reached an agreement that our code was working and correct for a few configurations, we executed all the configurations on the bridges-2 system for consistency on the V100 GPU in a single batch.

3 Experimental Results

First, we did hyperparemter tuning on the MNIST Fashion dataset [9]. It's a grayscale image dataset with 10 labels. Dataset contains 60,000 images for the training set and 10,000 images for the test set. We initially wanted to use CIFAR-10, but realized that would make the hyperparameter optimization very slow and resource intensive, since it requires deeper networks [10]. As oppposed to the MNIST digit dataset, we chose MNIST fashion as it seems quick to train, yet slightly difficult to optimize compared against the digit dataset. We wanted to see interesting results. Running locally on our laptops took around 1 hour for one hyperparameter tuning experiment to end. On the V100, it took just over an hour to run *all 6* configurations. We did 20 trials for each experiment with concurrency set to 20. We wanted to achieve peak utilization to efficiently use the GPUs, but weren't sure what a right number would be for concurrency. We tried 10 and 100, but found that 100 will hit OOM and fail a lot of experiments. So we chose 20 as a safe choice. Table 1 shows a summary of our results.

Table 1: Hyperparameter Summary for MLP on Fashion MNIST

HPO Tuner	Trial	Validation Accuracy	Learning Rate	Momentum	Batch Size	Feature Size
TPE	9	88.26%	0.0763	0.5338	–	–
Evolution	13	87.98%	0.0269	0.5765	–	–
Hyperband	2	88.33%	0.0855	0.6975	–	–
TPE with batch	6	88.26%	0.0476	0.4535	64	512
Evolution with batch	2	87.98%	0.0794	0.4306	32	512
Hyperband with batch	6	88.14%	0.0244	0.7010	32	512

3.1 MLP HPO Experiments with Learning Rate and Momentum

Figure 1 shows the result of the MLP with TPE. We observe that Trial 9 has the most optimal parameters with a validation accuracy of 88.26%. The optimal parameters for this trial are a learning rate of 0.0763 and a momentum of 0.5338. We also observe that the duration for all trials is about 5 minutes.

Figure 2 shows the result of the MLP with Evolution. We observe that Trial 13 has the most optimal parameters with a validation accuracy of 87.98%. The optimal parameters for this trial are a learning rate of 0.0269 and a momentum of 0.5765. We also observe that the duration for all trials is about 5 minutes.

Figure 3 shows the result of the MLP with Evolution. We observe that Trial 2 has the most optimal parameters with a validation accuracy of 88.33%. The optimal parameters for this trial are a learning rate of 0.0855 and a momentum of 0.6975. We also observe that the duration for all trials is also about 5 minutes.

For all of the tuners, there weren't any surprising differences. TPE turned out to perform the best, and all of the trials ran for the same amount of time, which makes sense as we're just tuning learning rate and momentum.

3.2 MLP HPO Experiments with Learning Rate, Momentum, Feature Size, and Batch Size

Figure 4 shows the result of the MLP with TPE. We observe that Trial 6 has the most optimal parameters with a validation accuracy of 88.26%. The optimal parameters for this trial are a learning rate of 0.0476 and a momentum of 0.4535. The optimal batch size is 32 and the number of features is 512. Duration for each trial ranged from 4-24 minutes. Trial 6 had a execution time of 5:43.

Figure 5 shows the result of the MLP with Evolution. We observe that Trial 2 has the most optimal parameters with a validation accuracy of 87.98%. The optimal parameters for this trial are a learning rate of 0.0794 and a momentum of 0.4306. The optimal batch size is 32 and the number of features is 512. Duration for each trial ranged from 4-21 minutes. Trial 2 had a execution time of 5:01.

Figure 6 shows the result of the MLP with Evolution. We observe that Trial 6 has the most optimal parameters with a validation accuracy of 88.14%. The optimal parameters for this trial are a learning rate of 0.0244 and a momentum of 0.7010. The optimal batch size is 64 and the number of features is 512. Duration for each trial ranged from 3-33 minutes. Trial 6 had a execution time of 5:01.

For all of the tuners, we are able to see speed differences now. TPE still performs the best, but it has a larger batch size than the others. It also has a longer time, which doesn't intuitively make sense.

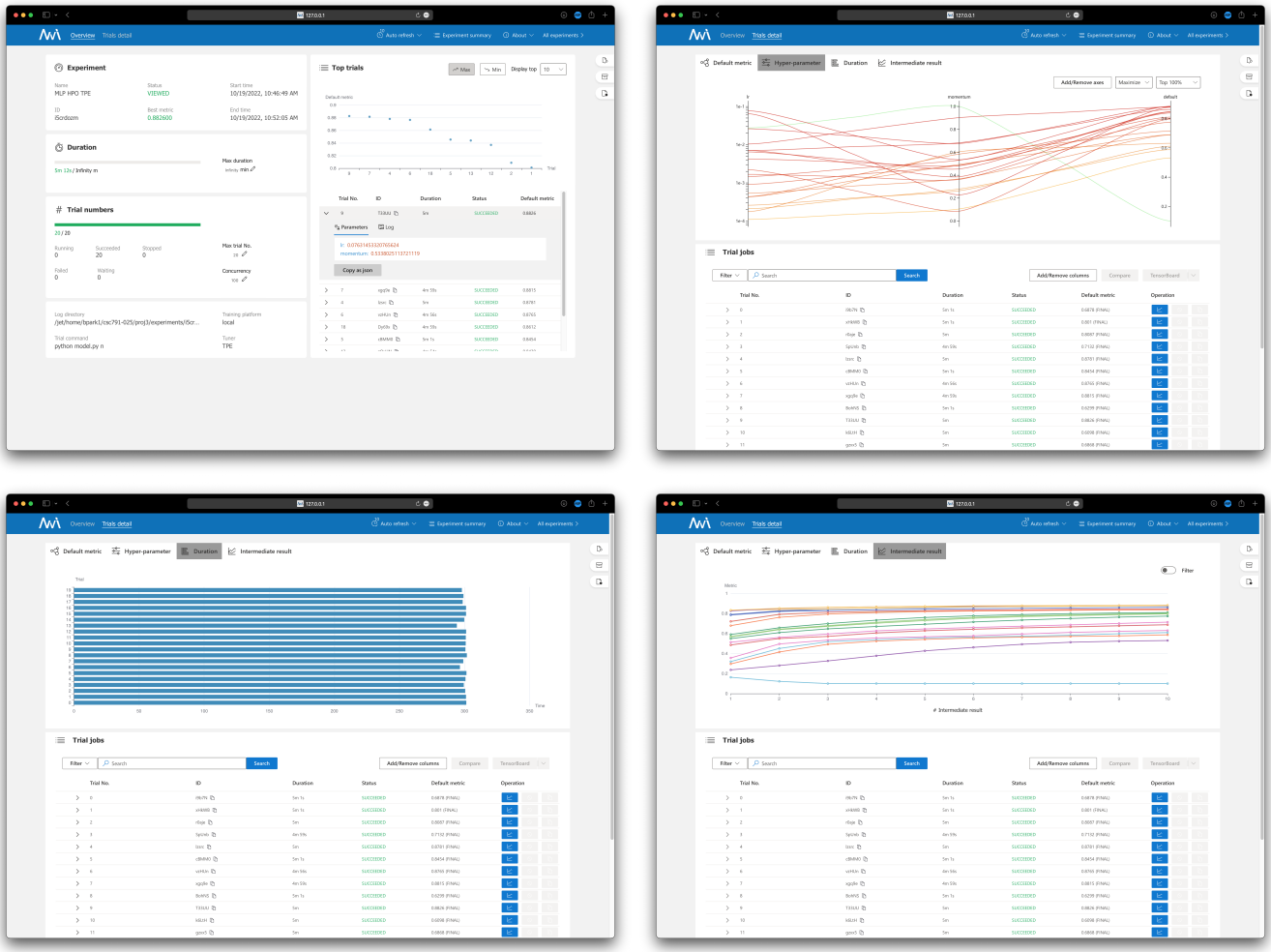

Figure 1: MLP with TPE Tuner on Learning Rate and Momentum

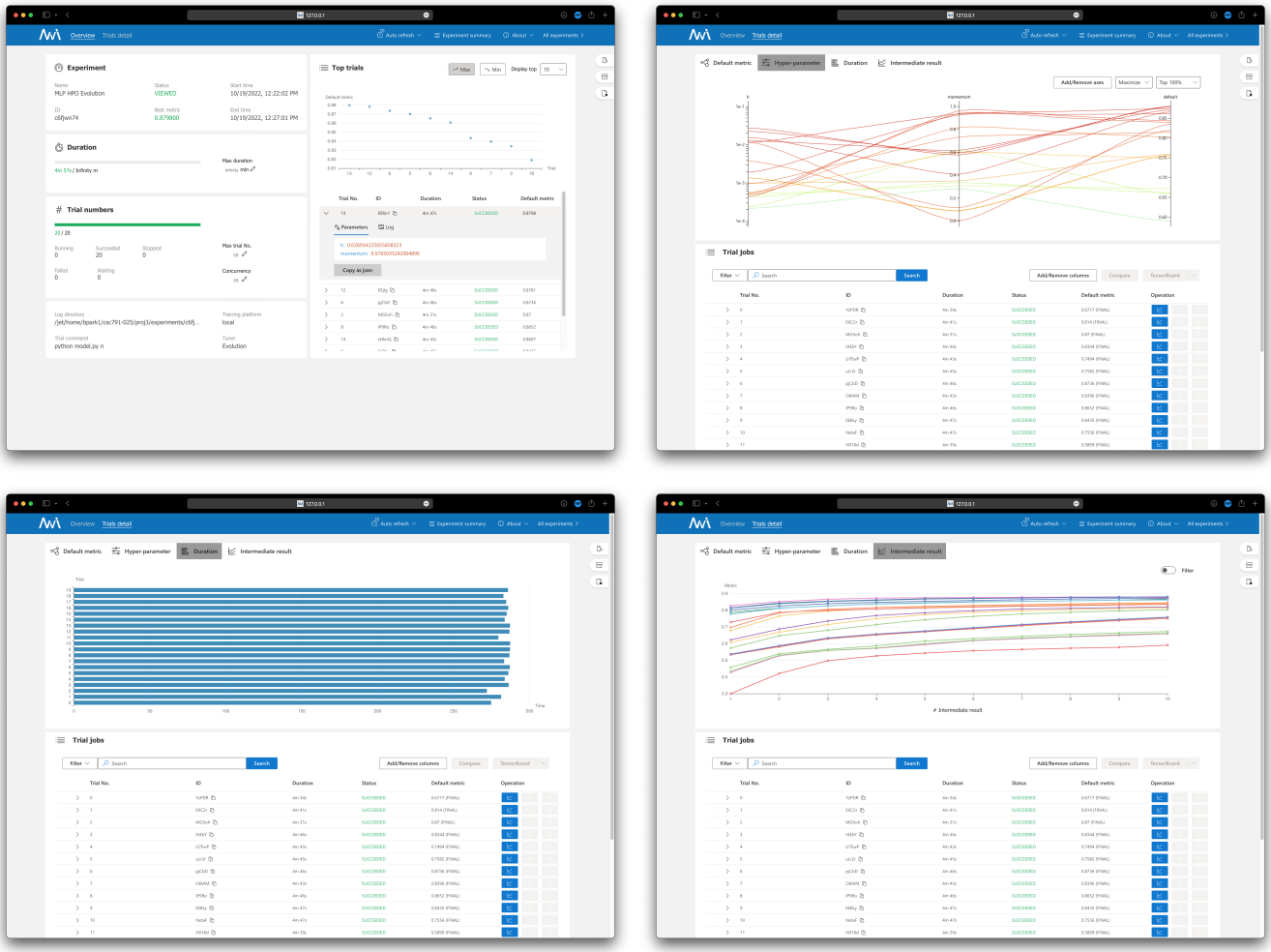

Figure 2: MLP with Evolution Tuner on Learning Rate and Momentum

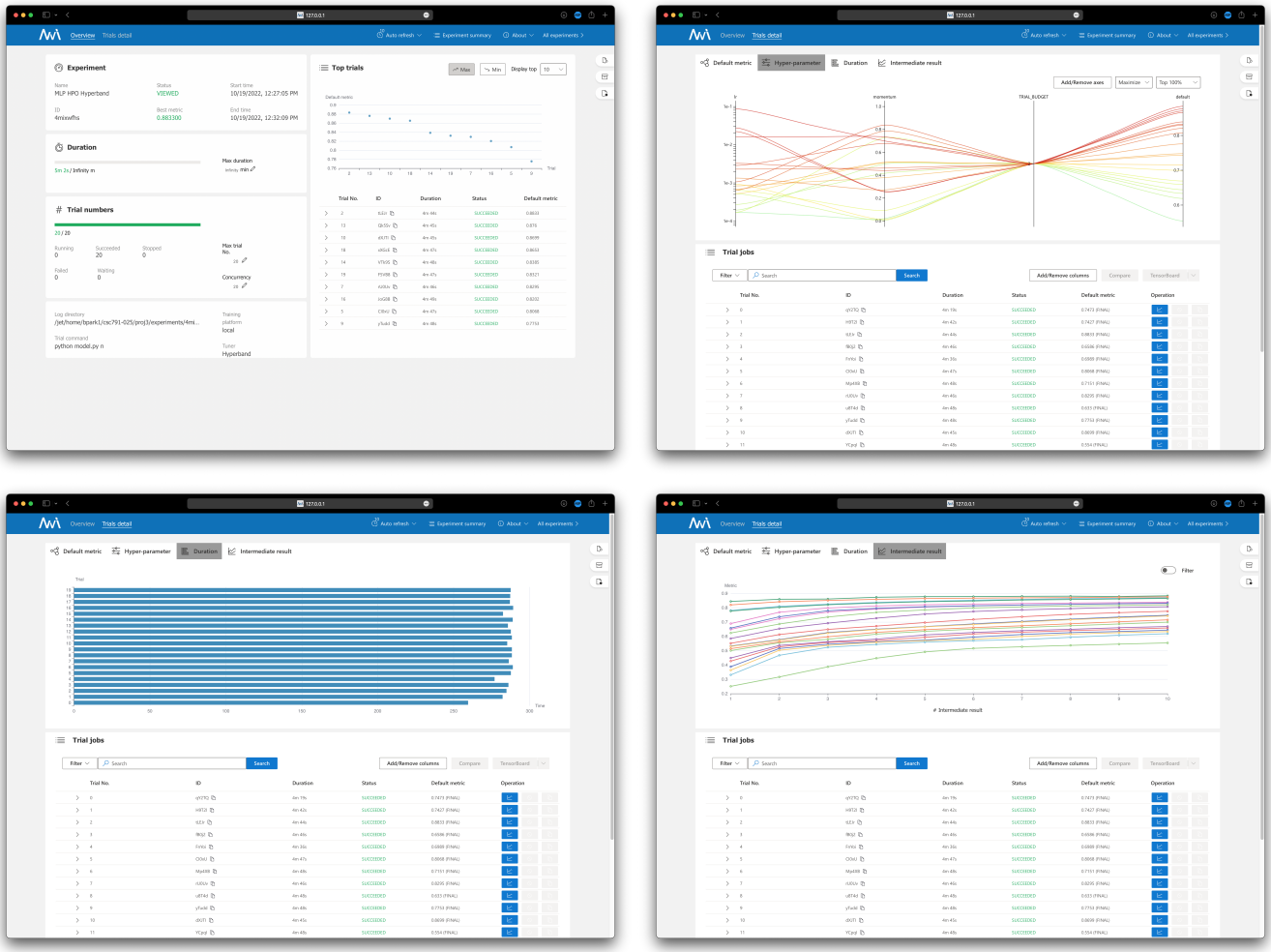

Figure 3: MLP with Hyperband Tuner on Learning Rate and Momentum

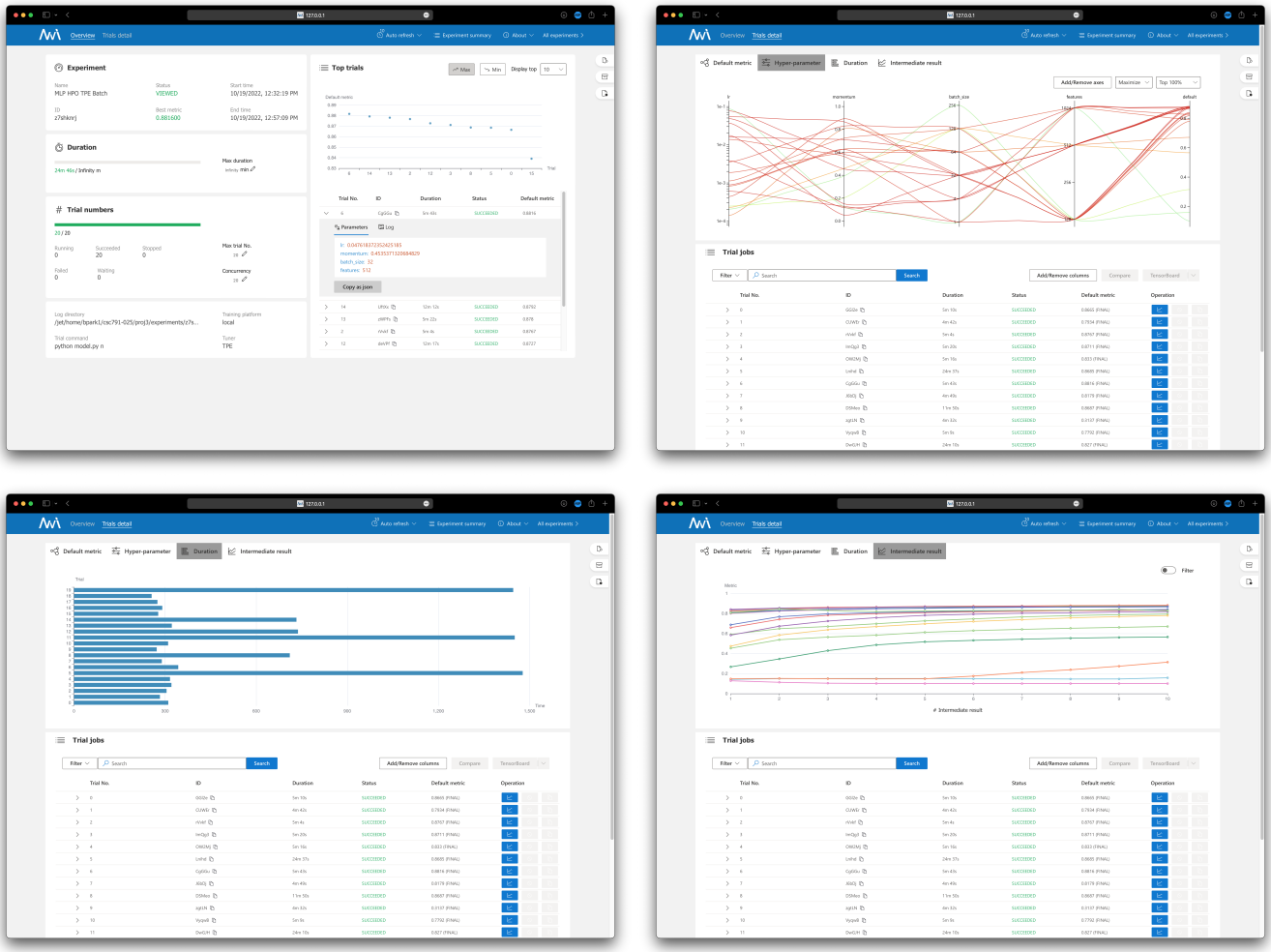

Figure 4: MLP with TPE Tuner on Learning Rate, Momentum, Feature Size, and Batch Size

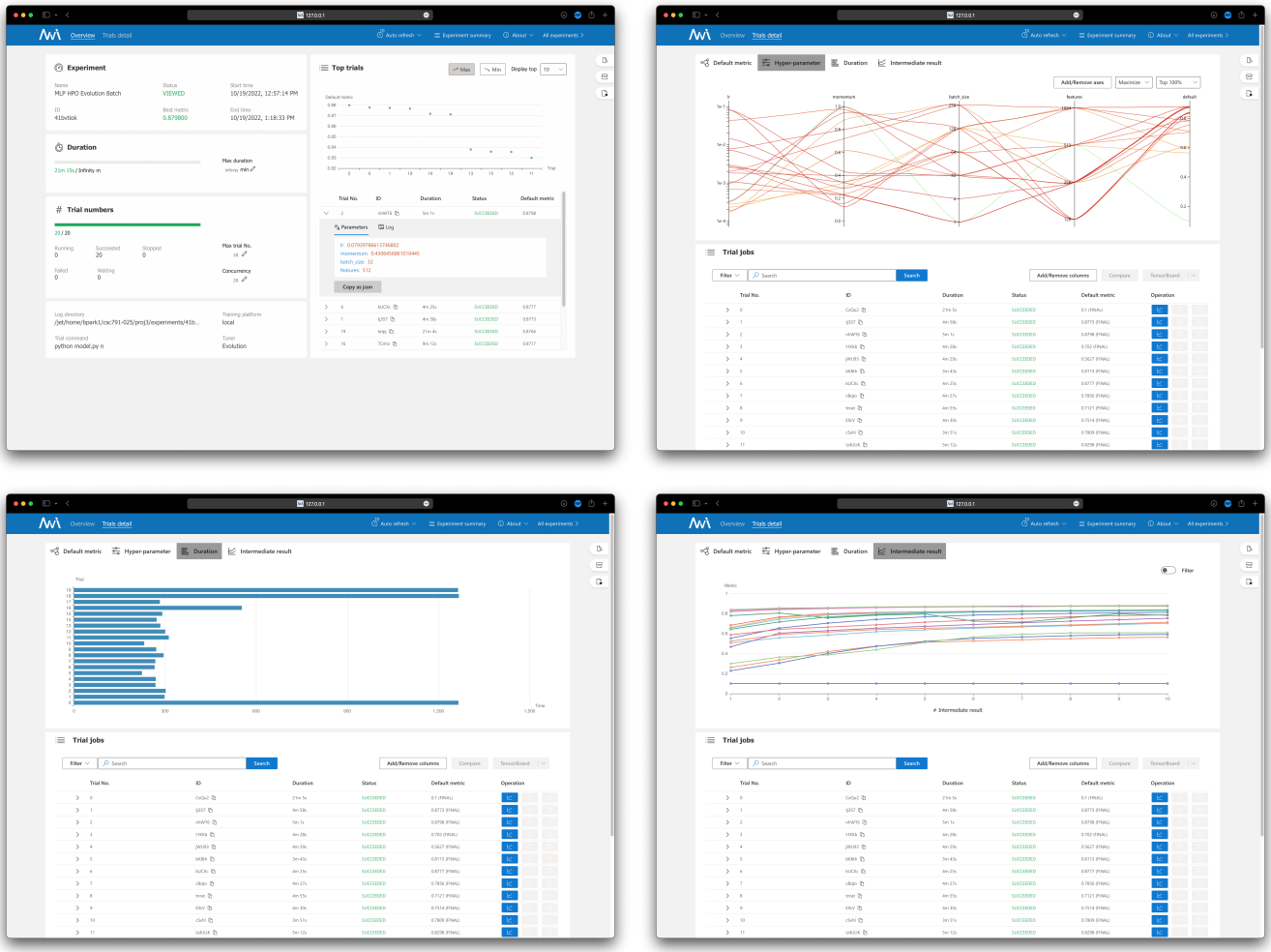

Figure 5: MLP with Evolution Tuner on Learning Rate, Momentum, Feature Size, and Batch Size

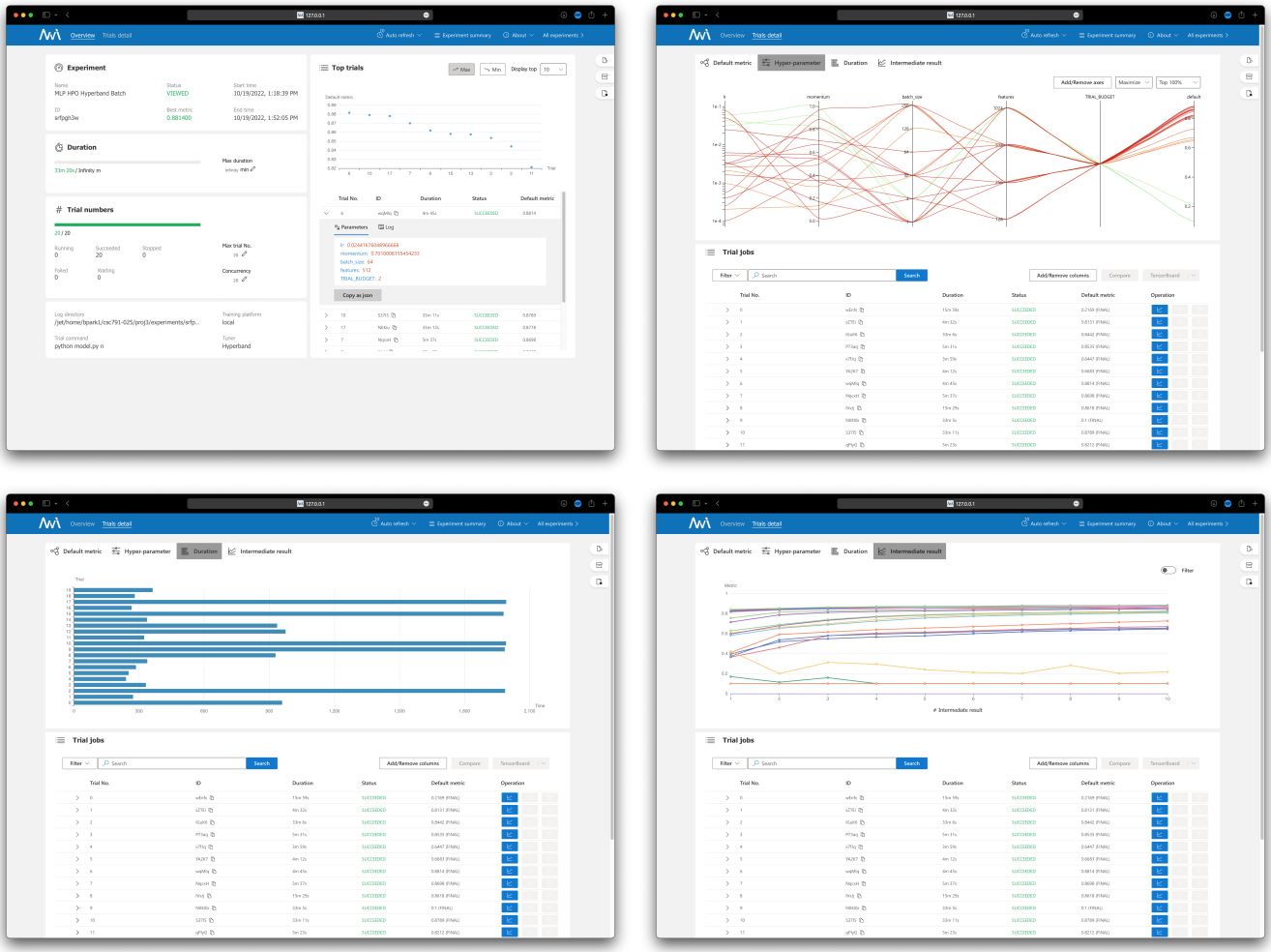

Figure 6: MLP with Hyperband Tuner on Learning Rate, Momentum, Feature Size, and Batch Size

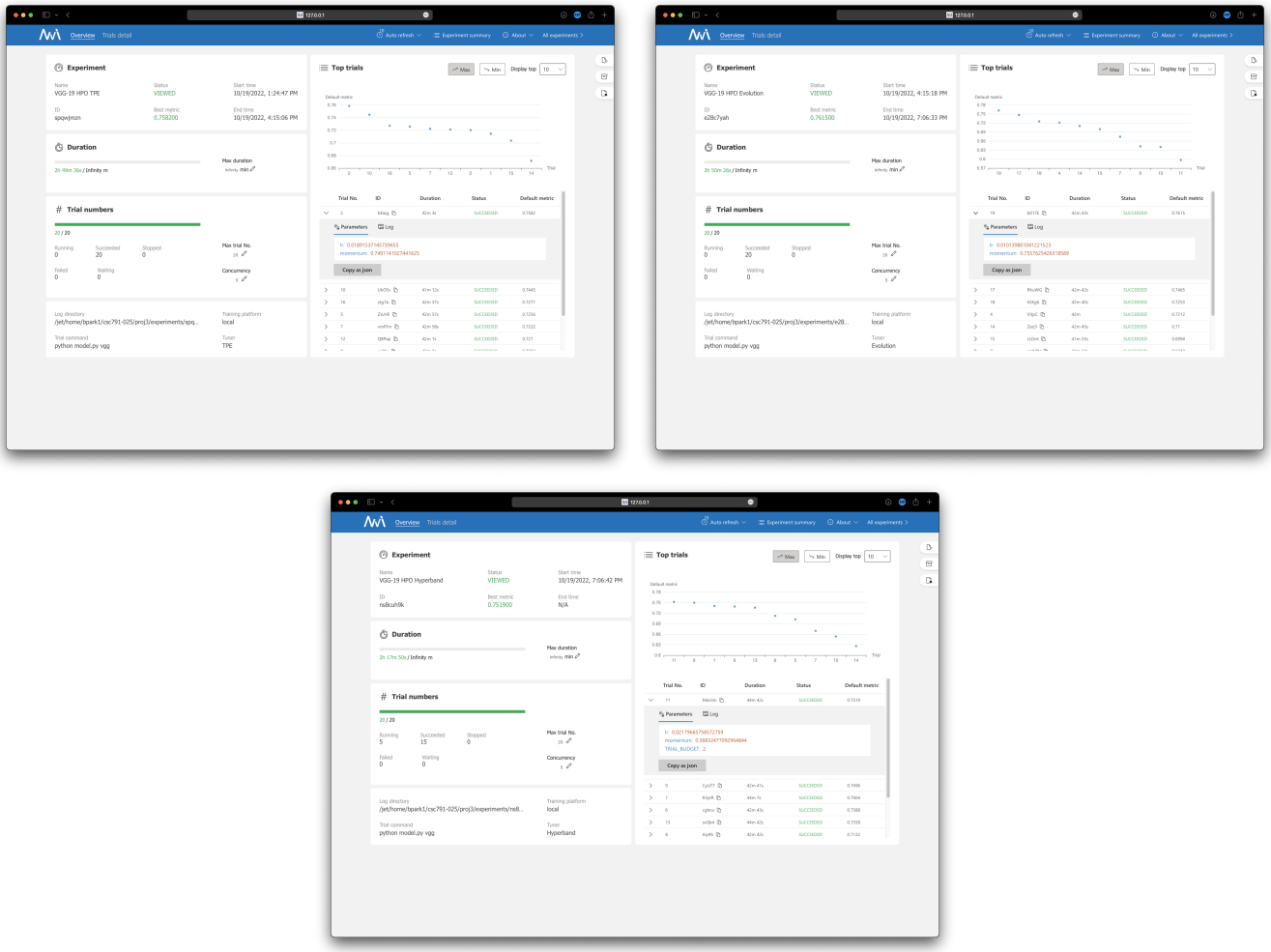

Figure 7: VGG HPOs on Learning Rate and Momentum

3.3 VGG-19 HPO Experiments with Learning Rate and Momentum

Just in case MLP didn't show interesting results, we also ran a few hyperparameter tuners for VGG-19. We couldn't run the whole suite of configurations as it took very long, so we capped our trials at 5 hours. Figure 7 shows a overall view of our results. Hyperband HPO tuner could not finish, so their results could've been better if ran to completion for a fair evaluation. Evolution HPO tuner had the best hyperparameters with a validation accuracy of 76.15%. We actually crashed our initial run, because it hit OOM. Lowering down the concurrency from 20 to 5 resolved this issue, as VGG-19 is a much larger model and the NVIDIA V100 has only 32GB of HBM memory.

4 Lessons Learned

NNI fortunately parallelizes the hyperparameter tuning process by enabling concurrent models to be trained at the same time. Unfortunately, we weren't aware of this when switching between hardware when running experiments. Thus we often crashed our own devices or had our devices struggle to operate due to a configuration suited towards high end GPU like the V100.

There are actually many competing hyperparameter optimizers and tuners out there that we were aware of. PyTorch presents Ray Tune in their documentation as a recommendation for hyperparameter tuning [11]. There is also an issue and discussion about what the differences between the two are, which is positioning. I think the main difference is that NNI is a platform that also eases for other methods like pruning and quantization. So much more hyperparameter tuning frameworks exist like HyperOpt, sklearn's GridSearchCV, Optune, Weights & Biases, and many more [12]. Some support classical machine learning algorithms, and other can support deep neural networks. But in the end, the tools seem to all serve a similar purpose with different positioning.

5 GitHub Repository

The GitHub repository for this report is publicly accessible here [13]. To reproduce our findings, please read the `README.md` under the `proj3` directory. If there are any setup issues on bridges-2 supercomputer or locally, please contact bcpark@ncsu.edu.

References

[1] Microsoft, "Neural Network Intelligence," 1 2021.

[2] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization," in *Advances in Neural Information Processing Systems* (J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K. Weinberger, eds.), vol. 24, Curran Associates, Inc., 2011.

[3] E. Real, S. Moore, A. Selle, S. Saxena, Y. I. Leon-Suematsu, Q. V. Le, and A. Kurakin, "Large-scale evolution of image classifiers," *CoRR*, vol. abs/1703.01041, 2017.

[4] L. Li, K. G. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar, "Efficient hyperparameter optimization and infinitely many armed bandits," *CoRR*, vol. abs/1603.06560, 2016.

[5] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2014.

[6] "Arc: A root cluster for research into scalable computer systems." `https://arcb.csc.ncsu.edu/~mueller/cluster/arc/`.

[7] "Nvidia v100 datasheet." `https://images.nvidia.com/content/technologies/volta/pdf/volta-v100-datasheet-update-us-1165301-r5.pdf`.

[8] S. T. Brown, P. Buitrago, E. Hanna, S. Sanielevici, R. Scibek, and N. A. Nystrom, "Bridges-2: A platform for rapidly-evolving and data intensive research," in *Practice and Experience in Advanced Research Computing*, PEARC '21, (New York, NY, USA), Association for Computing Machinery, 2021.

[9] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," 2017.

[10] A. Krizhevsky, "Learning multiple layers of features from tiny images," pp. 32–33, 2009.

[11] R. Liaw, E. Liang, R. Nishihara, P. Moritz, J. E. Gonzalez, and I. Stoica, "Tune: A research platform for distributed model selection and training," *CoRR*, vol. abs/1807.05118, 2018.

[12] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," in *Proceedings of the 25rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019.

[13] "Csc 791-025 github repository." `https://github.com/briancpark/csc791-025/tree/main/proj3`.