HyperTuner: Visual Analytics for Hyperparameter Tuning by Professionals

Tianyi Li*
Computer Science Department
Virginia Tech

Gregorio Convertino†
User Experience Design
Cloudera

Wenbo Wang ‡
User Experience Design
Cloudera

Haley Most §
Cloudera Data Science Workbench
Cloudera

Tristan Zajonc ¶
Cloudera Data Science Workbench
Cloudera

Yi-Hsun Tsai ‖
Cloudera Data Science Workbench
Cloudera

ABSTRACT

While training a machine learning model, data scientists often need to determine some hyperparameters to set up the model. The values of hyperparameters configure the structure and other characteristics of the model and can significantly influence the training result. However, given the complexity of the model algorithms and the training processes, identifying a sweet spot in the hyperparameter space for a specific problem can be challenging. This paper characterizes user requirements for hyperparameter tuning and proposes a prototype system to provide model-agnostic support. We conducted interviews with data science practitioners in industry to collect user requirements and identify opportunities for leveraging interactive visual support. We present HyperTuner, a prototype system that supports hyperparameter search and analysis via interactive visual analytics. The design treats models as black boxes with the hyperparameters and data as inputs, and the predictions and performance metrics as outputs. We discuss our preliminary evaluation results, where the data science practitioners deem HyperTuner as useful and desired to help gain insights into the influence of hyperparameters on model performance and convergence. The design also triggered additional requirements such as involving more advanced support for automated tuning and debugging.

Index Terms: Human-centered computing—Visualization—Visualization techniques—Hyperparameter Tuning; Human-centered computing—Visualization—Visualization design and evaluation methods

1 INTRODUCTION

The increasing availability of big data is invigorating a more prevalent use of machine learning (ML) models among a wide variety of users to solve real-world problems. As the demand for application-specific ML models increases, tools to enable users to efficiently and confidently build ML models have become increasingly important. Recent research in the VIS community has highlighted various challenges experienced in the design and application of ML models that call for better visual analytics support in the current tools [25].

Building a suitable ML model is an expensive and iterative process. It includes data preparation, feature engineering, model implementation, hyperparameter tuning, debugging, validation, and other project-specific tasks. The same model algorithm often needs different hyperparameter settings when training on different datasets or serving different analysis goals. In many deep learning models, hyperparameters such as the number of layers or the dropout rate can dramatically affect performance.

Prior visual analytics research on supporting the model tuning focuses on the following three aspects. 1) Visualizing model structures with prior knowledge [17,28]. This can be very useful to understand the model components and their relationships so that data science practitioners make educated decisions on hyperparameter tuning. The disadvantage is that such visualization is developed for one specific model and barely generalizable to a different model. 2) Visualizing model predictions with additional interpretable models [13,23]. This approach is data-oriented and model-agnostic, learning a local interpretable model around the predictions to explain ML models. This can help data scientists reverse-engineer the model to detect which part is not performing well and conduct targeted hyperparameter tuning. It is generalizable across models but expensive to train the additional local models and have to be applied to one ML model at a time. 3) Algorithmic approaches [6, 30]. Automatic hyperparameter tuning like surrogate-based optimization takes advantage of the growing capability of computing infrastructures and higher evaluation budgets. However, while automated hyperparameter tuning was a key feature requested by data scientists we interviewed, they also expressed a need to develop intuition through visualization of experiments.

To understand the empirical hyperparameter tuning process, we conducted interviews with six data science practitioners in industry who work in different job roles. We learned that data science practitioners usually spend a considerable amount of time manually tuning hyperparameters, taking notes on paper notebooks or building their own visualizations due to the limited visual analytics support. We collected the common practices, user needs, and pain points in the interviews, and characterized the findings as a workflow of the key steps. We identify the sub-steps and leverage points, and demonstrate the corresponding designs via a prototype. We evaluated the prototype with the same group of data science practitioners, who provided positive and constructive feedback for improvement. The paper concludes with the result of this first evaluation and future directions of improvement and extension.

The contributions of this work are listed as follows:

1. Analyzed real-world user practices and needs for effective hyperparameter tuning.
2. Formalized the hyperparameter tuning process as a workflow.
3. Implemented and evaluated a prototype to support hyperparameter tuning.

Consistent with the themes of the workshop, this research sits at the intersection of the Machine Learning (ML) and Visual Analytics. It focuses on empowering data science practitioners via new types of user interactions and visual analytics tools. In particular, it aims at guiding and advancing the development of visual analytics tools for interactive hyperparameter tuning.

*e-mail:tianyili@vt.edu
†e-mail:gconvertino@cloudera.com
‡e-mail:wenbo@cloudera.com
§e-mail:haley@cloudera.com
¶e-mail:tristanz@cloudera.com
‖e-mail:yihsuntsai@cloudera.com

2 Problem in Context

The set of problems we focus on in this paper appears in the context of training a machine learning model to answer some questions regarding a dataset. For example, given all the customers of a company in the past 10 years, who are likely to extend their subscriptions, and who might cancel and leave? When building an ML model to make predictions on customer churn, the data science practitioners first acquire, clean, and enrich the customer data (Step 1 in the top half of Figure 1). Next, they engineer the relevant features of the customers to be used in making predictions (Step 2 in the top half of Figure 1). They also need to select what ML model to use: more traditional models like a decision tree, or some complicated neural network models? (Step 3 in top half of Figure 1). While training the selected model, many models have hyperparameters that must be set and tuned. For example, a decision tree needs a criterion function (Gini index or entropy) to measure the quality of a node split, a maximum depth of the tree, and other hyperparameters. A neural network can have many hyperparameters such as the number of layers, which activation function to use, and so on (Step 4 in the top half of Figure 1). With the hyperparameters set, the models are trained and validated with a subset of the customer data (Step 5 in the top half of Figure 1). If not satisfied, the data science practitioners iterate by returning to any of the prior four steps, depending on the results. Finally, when satisfied, they can share the trained model with other data analysts and domain experts or embed the model into a business process.

We focus on the problem of hyperparameter tuning: see steps 4 and 5 and the bottom half of Figure 1. It's worth prefacing, first, the distinction between two types of parameters. The first is the *hyperparameters*, which are set by the users before the training process begins. The second is the *learned parameters*, for example, "weights" of the data features the model learns from the dataset.

At the bottom half of Figure 1, we indicate the three leverage points of prior research to support hyperparameter tuning and, in this context, our focus in this paper. The first is to visualize the model structure (e.g., what layer of the neural network model is processing what data features [18]; see the vertical line with label 1). The second is to learn a local model around the model prediction (e.g., what features an image-recognition model captured to classify a specific image [23]; see the vertical line with label 2). The third is to algorithmically optimize hyperparameter values with an objective function (see the vertical line with label 3). In this work, we treat a ML model as a black box, and combine the second and third leverage points via interactive visual analytic to support hyperparameter tuning (see the vertical line with label 4).

3 Related Work

Hyperparameter tuning is an essential but time-consuming step in model training. In order to get an ML model to work well in practice, the hyperparameters need to be tuned when training the model and the best settings usually differ for different datasets. Revising the configuration of existing learning paradigms can sometimes lead to more improvement in model performance than inventing new ones [11, 20]. However, evaluating the effect of a given hyperparameter setting is expensive since it usually requires the model to be trained and tested, which is often computationally costly and may have random factors that make the evaluation even more difficult.

3.1 Visual Analytics Support in Parameter Search

Exploring the relationships between several explanatory variables and one or more response variables has been a prevalent challenge in many different application domains, including meteorology [21], biology [22] and medical science [5]. Statistical approaches such as the response surface methodology (RSM) [8] has been widely employed in developing solutions to those problems, and is further developed into the "design and analysis of computer experiments", and "visual parameter space exploration" [26] in the visual analytics community. Assessing the optimality of a black-box model often involves visualizing and sampling from multi-dimensional spaces, using surrogate and/or simulation models, contrasting the trade offs of several input settings, as well as understanding the influence of the inputs on the outputs. With these shared challenges, the frameworks and strategies developed for non-ML applications are generally applicable to hyperparameter tuning as well. For example, the four navigation strategies proposed by Sedlmair et al. [26] can also serve most of the exploration and analysis needs in the hyperparameter tuning process. However, hyperparameter tuning optimizes a machine learning model by steering the initial setting before the training process. This is different from optimizing learned parameters based on the data. Take a simple polynomial model, $y = ax^k + b$, as an example. The value of parameter a and b are adjusted to fit a given dataset, while the k is preset by the user to define the model structure. If the model performance is consistently bad for a certain k value, more tuning will be needed to determine a more appropriate k value. Unlike tuning learned parameters as in non-ML problems, users are required to have considerably abundant knowledge about the model algorithm in order to understand the relationship between the hyperparameter setting and the training process, in order to explore the influence of the hyperparameters to the model performance.

Prior work on hyperparameter tuning, or hyperparameter search, can be broadly divided into two camps: automated approaches developed by the Machine Learning or AI researchers (e.g., see review in [10]) and human-in-the-loop approaches developed by researchers working between Machine Learning and Visual Analytics (e.g., see review in [24]), as the one proposed in this paper. Below we review these two camps of research.

3.2 Automated Approaches

Claesen et al. [10] refer to the vision of a fully automated, self-configuring learning strategy, including hyperparameter search, as still the "holy grail of machine learning". Nevertheless, automatic approaches are showing increasing success on tuning hyperparameters of some models [6, 30]. Important contributions of the work done in this camp includes the formalization of key concepts (e.g., hyperparameter) [9], the identification of model tuning tasks that can be automated, and consequently software modules that implement specific hyperparameter optimization methods, such as Scikit-learn [19], or packages that focuses on Bayesian methods (e.g., see https://bayesopt.github.io/), such as Hyperopt [7] and ParamILS [16]. On the commercial end of this camp, sample software tools include https://cloud.google.com/automl/ and https://www.datarobot.com/.

While there are important advancements in the automated approaches camp, it's important to recognize that there are many practical scenarios where such automatic approaches are not applicable. An exhaustive *grid search* is computationally expensive and time-consuming; *random search* and surrogate-based model optimization require enough trials for a given dataset. More advanced sampling techniques such as latin hypercubes and low-discrepancy sequences also suffer from the same problems. Furthermore, implementing automatic approaches usually requires writing additional scripts with an in-depth understanding of the model and search algorithms, which excludes non-expert users of ML models or novice practitioners. As we learned in the interviews, the tuning process is either highly inefficient with manually keeping a notebook, or relies on automatic optimization algorithms that are usually expensive to implement without guarantee of success (convergence). Even when automated approaches are used, data scientists expressed a desire to combine both automated and manual iteration in one framework to help keep track of experiments and develop some intuition for potential modeling avenues to explore. In this work, we investigate where and how to leverage human guidance to improve the hyperparameter tuning efficacy.

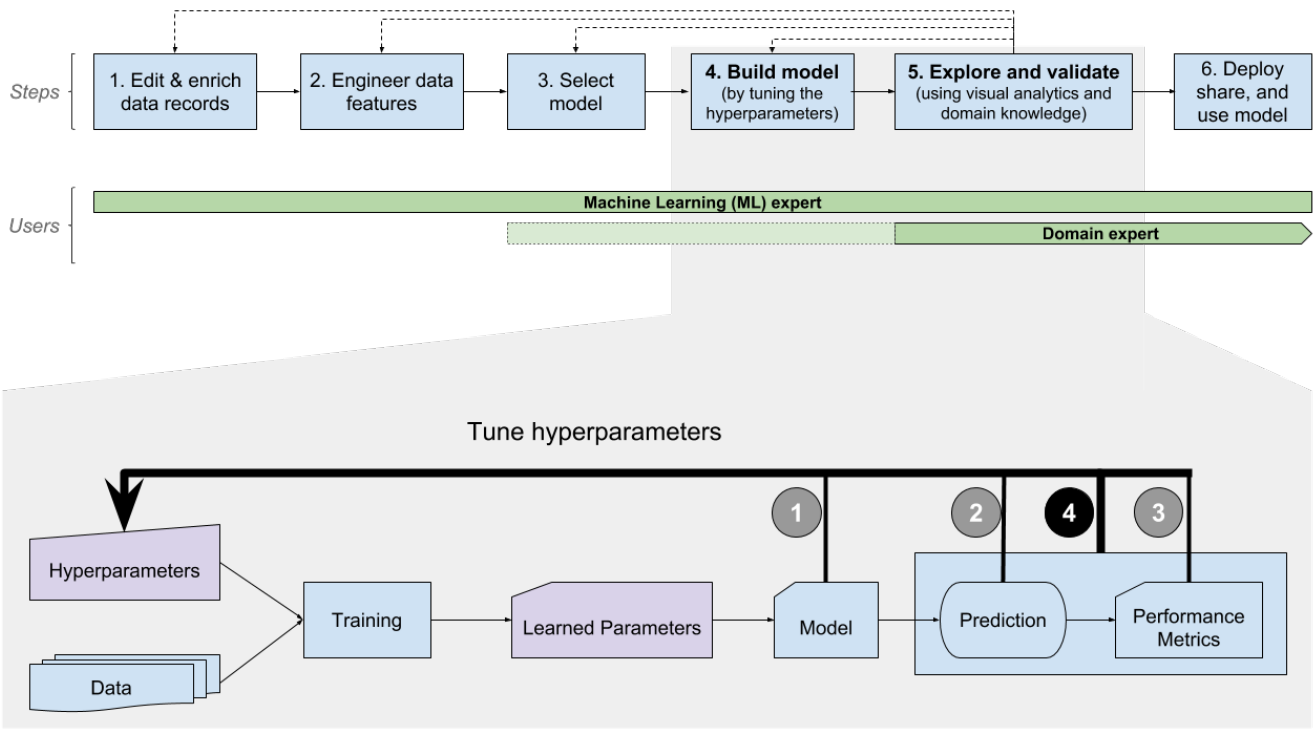

Figure 1: Problem in context. The top half (context): overall work process by a data science practitioner of addressing hard questions by applying an ML model to a dataset [24], p. 642. The bottom half (problem): model building via hyperparameter tuning and result validation.

3.3 Human-in-the-loop Approaches

The current reality of hyperparameter tuning is highly human-driven. Hyperparameter tuning is usually performed manually [9] following rules-of-thumb and experience accumulated through practice [14, 15]. Especially with complex models, the current trial-and-error process is inefficient in terms of the time spent and the computational load [22], and does not favor reproducibility, knowledge transfer, and collaboration (e.g., [10]). To address these limitations, as well as increasing the transparency of ML models to humans, current research efforts in the human-in-the-loop camp have focused on three visual analytics foci: model structure, model prediction, model performance metrics.

Visualizing model structure. Many visualization techniques have been developed to help users understand the structure of different ML models. Liu et al. [17] developed a visual analytics system that shows all the layers, neurons, and other components of the convolutional neural networks, as well as how the training data is processed within. GANViz [29] visualizes the adversarial training process of generative adversarial nets with linked coordinated visualizations. Rather than devoting to an in-depth understanding of a specific model, we investigate a more light-weight and general-purpose support for model-agnostic hyperparameter tuning.

Interpreting model prediction. There is an evolving research interest in model-agnostic interpretation, that focuses on understanding model prediction behaviors. Riberio et al. developed LIME [23], a model-agnostic approach that learns an interpretable model locally around the prediction. This framework is generalizable to different models and dataset, and efficiently enhances the interpretability of a given model. However, before drilling down into such expensive interpretation and copious details, we focus on comparing many alternative models and identifying better ones.

Interpreting model performance metrics. The lack of support for evaluating the performance of ML models has been known for at least a decade now. For example, Patel and collaborators [12] in a 2008 study with data scientists observed that tuning ML models is an iterative and exploratory activity. It was hard for the data scientists in the study to track performance across iterations, which makes tuning process challenging. They argued that ML tools should include more suitable visualizations to help data scientists with their work. A recent attempt to provide more suitable visualizations of model evaluations metrics and model comparisons was made by Tsay and colleagues [27]. This is a research area that we expect will receive increasing attention in the near future.

4 Hyperparameter Tuning Requirements

In the first phase of the project, we interviewed data science practitioners in industry to investigate their hyperparameter tuning practice, as ML experts and as colleagues of domain experts with business needs and knowledge. In this section, we characterize the key steps of the process and user needs not yet addressed by current tools.

4.1 Method

We interviewed six data science practitioners. While all were experienced in hyperparameter tuning, they held various job roles, including "data scientist", "data science engineer and researcher", "machine learning engineer", and "software engineer for a data analytics product". We will refer to the six interviewees as P1, P2, P3, P4, P5, P6

4.2 Hyperparameter Tuning: Practice

Our interviews may be affected by potential sampling biases: the interviewees were based in the United States and the UK; they worked for a software company that builds applications for machine learning and analytics on big data. However, the sample included a good variety of job roles and, we believe, can represent the need of a broader range of data science practitioners in the industry (i.e.,

gru_width	seqlen	layer_width	num_layers	w2v_size	learning_rate	optimizer	gru_depth	history_size	separate_history_model	batch_size	final_val_loss	name	best_val_loss	num_epochs
128	64	(256,)	1	300	0.0005	<class 'keras.optimizers.Adam'>	1	5	False	64	0.155191	hobnailed_capitals		
128	64	(128,)	1	300	0.0005	<class 'keras.optimizers.Adam'>	2	5	False	64	0.151965	plunk_ferociousness		
256	64	(256,)	2	300	0.0005	<class 'keras.optimizers.Adam'>	2	5	False	64	0.205403	crumbles_conveyances		
128	64	(256,)	1	300	0.0005	<class 'keras.optimizers.Adam'>	1	5	False	64	0.153475	synergistic_athlete		
256	64	(256,)	2	300	0.0005	<class 'keras.optimizers.Adam'>	2	5	False	64	0.160458	texan_wetnesss	0.146566	18
128	64	(128,)	1	300	0.0005	<class 'keras.optimizers.Adam'>	2	5	False	64	0.166863	extremely_corneal		
256	64	(256,)	2	300	0.0005	<class 'keras.optimizers.Adam'>	2	5	False	64	0.160458	texan_wetnesss	0.146566	18
256	64	(256,)	2	300	0.0005	<class 'keras.optimizers.Adam'>	2	5	False	64	0.160458	texan_wetnesss	0.146566	18
256	64	(256,)	2	300	0.0005	<class 'keras.optimizers.Adam'>	2	5	False	64	0.147994	lamed_sweltered	0.142187	19
256	64	(256,)	2	300	0.0005	<class 'keras.optimizers.Adam'>	2	5	False	64	0.142559	typist_cigarettes	0.139903	30
256	64	(256,)	2	300	0.0005	<class 'keras.optimizers.Adam'>	2	5	False	64	0.163339	taxonomies_rubier	0.145622	24
256	64	(256,)	2	300	0.0005	<class 'keras.optimizers.Adam'>	2	5	False	64	0.147242	tract_dehydrate	0.141203	63
256	64	(256,)	2	300	0.0005	<class 'keras.optimizers.Adam'>	2	5	False	64	0.182121	mirach_burroughs	0.143775	63
256	64	(256,)	2	300	0.0005	<class 'keras.optimizers.Adam'>	2	5	False	64	0.159622	organizations_reuses	0.139909	56
256	64	(256,)	2	300	0.0005	<class 'keras.optimizers.Adam'>	2	5	False	64	0.160787	theologians_tugging	0.137866	52
256	64	(256,)	2	300	0.0005	<class 'keras.optimizers.Adam'>	2	5	False	64	0.150513	tripods_strongly	0.140211	56
256	64	(256,)	2	300	0.0005	<class 'keras.optimizers.Adam'>	2	5	False	64	0.168799	yahoos_capering	0.139792	54
256	64	(256,)	2	300	0.0005	<class 'keras.optimizers.Adam'>	2	5	False	64	0.159466	email_invader	0.139116	53
256	64	(256,)	2	300	0.0005	<class 'keras.optimizers.Adam'>	2	5	False	64	0.154115	sperrys_semesters	0.142206	74
256	64	(256,)	2	300	0.0005	<class 'keras.optimizers.Adam'>	2	5	False	64	0.160458	developmental_furthered	0.146566	18

Figure 2: A text file showing some experiment history from P4

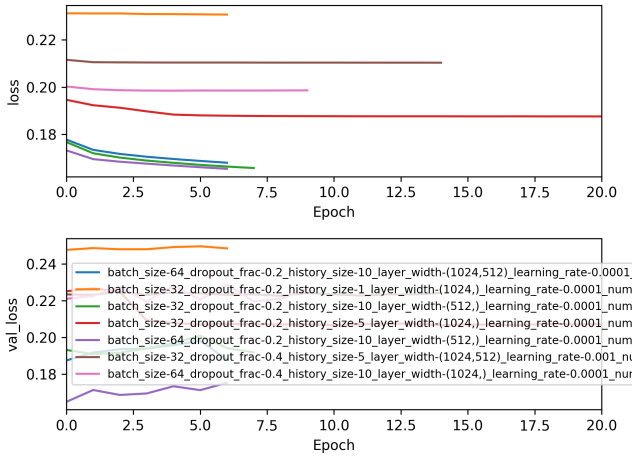

Figure 3: Example visualization from P4

their colleagues and customers in other companies). In this first phase of the project, we aimed at a qualitative characterization of the hyperparameter tuning process and user requirements rather than quantitative findings.

The procedure of each interview followed a semi-structured method and included two sub-sessions. The first sub-session, lasting about 15 minutes, investigated the job role and working context of the interviewee:

1. Please briefly describe your job role.

2. Please describe the most common tasks for your job role.

The second sub-session, lasting about 45 minutes, investigated the model tuning practices:

1. Please describe example projects, where you needed to train and tune models.

    - What was your dataset, model and hyperparameters, evaluation metrics?
    - What do you look at to decide if the model can be better? What and how to tune?
    - How many experiments do you typically compare?

2. How would you perform the above task with other models? (generalizability of 1)

3. What kind of assistance would help you with the tasks you described?

We asked follow-up questions depending on the interviewee's response. In addition, we asked the interviewees to introduce us to the tools they use and, when possible, provide us with examples of their projects (e.g., outputs from hyperparameter tuning in a representative project, see Figure 2 and 3). Each interview was recorded and transcribed.

Two of the co-authors performed a qualitative analysis on each transcript and summarized common themes (i.e., common steps, tools, and pain points). The themes were then validated with other two co-authors, who are data science domain experts.

4.2.1 How many tuning iterations or rounds do you do?

The interviewees referred to a model tuning task as a project involving multiple experiments, which may last several hours or days. Two of the six data scientists (P1 and P6) mentioned the strategy of starting from tuning simple models first (e.g., a logistic regression) and then, based on resources available (i.e., time and computation) vs. performance required, moving to tune more complex models. Generally, a model tuning project would require at least 20 experiments (P1) and might take as many as about 60 experiments in the case of a deep learning model (P4). This number could be significantly higher if parts of the tuning and experimental process were automated.

4.2.2 What hyperparameters do you tune? How?

The number of hyperparameters differs model by model. In the case of a deep learning model, there are often dozens of hyperparameters. The interviewees would focus on tuning *"a dozen or more" (P4)* hyperparameters. It usually relies on the data science practitioners to plan and *"keep everything as a changeable parameter or knob" (P4)* in their code.

The data science practitioner who shared the Figures 2 and 3 describes his typical hyperparameter tuning process below. Not all hyperparameter values need to be tuned, such as the learning rate. Some commonly tuned hyperparameters include the optimizing algorithm, dropout rate, the number of layers, the width of each layer, to name just a few. *"I do not really try to fix those right off the bat. Instead, I define limits, so I bound what values I think they could take, and then I either do a grid search or a random parameter search"*. In the example project P4 described to us, it usually takes about 6 hours to train a model with one set of hyperparameters. In addition, there's only a certain amount that can be parallelized due to the limited CPU resources. In such situations, random parameter search is more commonly used because the results would converge faster, and provide a better sense of the hyperparameter space.

When deciding if a specific hyperparameter should be tuned more, P4 reported that he usually holds *"all other features constant [across experiments], and just experiment with one."* If the results do not change much, this hyperparameter might not be worth further tuning. For example, some users find learning rate not very important to affect their results. It is useful to figure such things out sooner, to invest computation resources into tuning other hyperparameters.

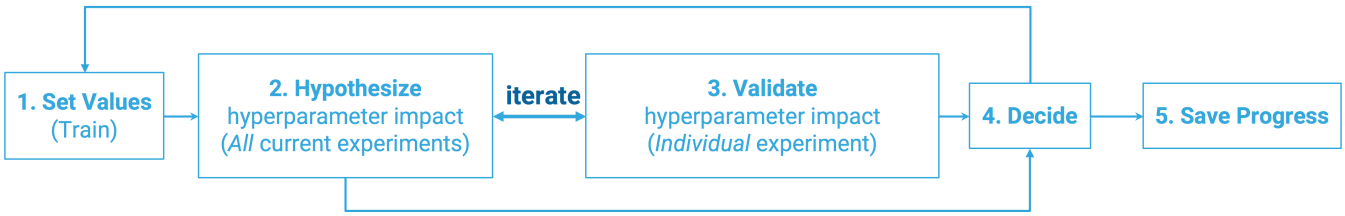

Figure 4: Formalized workflow of the hyperparameter tuning process.

4.2.3 What performance metrics do you track? How?

The commonly used performance metrics for supervised models include accuracy, precision, recall, and ROC curves. Other examples are learning curve (to check the slope and when it gets saturation), training loss vs. validation loss (to check when the latter increases as the first keeps decreasing to detect overfitting).

4.2.4 Workflow

The six data science practitioners pointed to a similar underlying process of hyperparameter tuning. What we learned was consistent with the reports from the literature about the general process, which we summarized at the top of Figure 1. An interviewee (P1) summarized the process as follows: *"We go through a typical data science workflow, which is to clean data, train a model, and then open it up over an API to a web front-end."*. We formalize the hyperparameter tuning process as a workflow with five sub-steps (shown in Figure 4), with the first four forming a loop.

Sub-step 1: Set hyperparameter values. At the outset of the workflow (the first square in Figure 4), the ML experts initiate the first batch of experiments by setting hyperparameter values based on their understanding of the data, the model algorithm, and the problem to solve. This sub-step reoccurs later as a restart of the loop if, after sub-step 4 (the fourth square in Figure 4), the ML expert decides that more tuning is still needed.

Sub-step 2: Hypothesize the impact of tuned hyperparameters with results of all experiments. At this stage the ML expert has just run a batch of experiments and wants to answer two questions: 1) What are the most impactful hyperparameters in these experiments? Is hyperparameter X relevant? 2) How do the hyperparameters influence the performance of the model and what performance metrics to consider? This sub-step is performed with the support of summative reports of the hyperparameters and performance metrics for a full batch of experiments.

Sub-step 3: Validate hypotheses with details of individual experiments. The ML expert may need to drill into the details of specific experiments to test the hypotheses developed in sub-step 2: 1) What do the details of this experiment say about my hypotheses? 2) Do I trust the predictions of this models by looking at the results? This sub-step represents an in-depth investigation that starts and ends back in sub-step 2 (see bidirectional arrow in Figure 4). It is performed with the support of detailed reports on hyperparameters and performance metrics from an individual experiment. Multiple micro-iterations occur between sub-steps 2 and 3, typically.

Sub-step 4: Decide if more tuning is needed. Once the ML expert has analyzed the results of the current batch of experiments, s/he needs to decide: 1) Does the (best) model performance meet my expectations? If not, 2) Will more training improve the model performance and will it be worth the efforts, given the resources? This sub-step is performed with the support of the summative and detailed reports from the prior two steps

Sub-task 5: Review and Save Progress. If the ML expert decides, in sub-step 4, that no more tuning is needed, then he answers these questions: 1) How well am I able to recall the tuning process and communicate its results? 2) How useful are the records of my tuning progress? What is missing? What is superfluous? This sub-step is performed with the support of a final project-level report summarizing all the experiments from all batches plus any comments or reminders the practitioner recorded during the tuning process.

4.3 Hyperparameter Tuning: Support Needed

Following the workflow in Figure 4 we identified at each step needs for visual analytics support that are not fully addressed by current data science tools.

4.3.1 Analytics of batches of experiments

One of the most evident needs emerging from the interviews is the need to aggregate results across experiments and conduct group-level comparisons among the experiments in a batch. The data science practitioners need visualizations that help determine which hyperparameters values are satisfying and which ones require more exploration. They also need to interactively customize the visualization. In the words of an interviewee (P2): *"Visualization is to bring human interpretation in hyperparameter tuning... build a visualization that is a natural interpretation of the actual data passed in, ... Users always want variation, they should be allowed to customize the visualization"*. Currently, these visualizations are created manually and in ad hoc fashion. A data science practitioner (P1) summarize his current practice as follows: *"You try different combinations of hyperparameters, keep track of the performance, and visualize it somehow"*.

4.3.2 Analytics of individual experiments

A second general need is a support to investigate results and metadata of an individual experiment (trained model). As shown in Figure 4, these investigations happen as drill-down analysis to test the current working hypotheses. For example, the user may need to understand the training history of an experiment and if the model predictions can be trusted. Additionally, s/he may want to review the metadata as a reminder of the hyperparameter values used. If the experiment is worth tracking for later comparisons, s/he can annotate some notes from the analysis. In particular, three of the six practitioners (P1, P4, P6) mentioned the need to review interpretability details such as examples of misclassifications by a supervised ML model. P1: *"Interpretability is also an important factor to track when turning hyperparameters. Is it predicting the right class for the right reason? I would want to get examples which get classified correctly or not."* [23].

With computationally expensive experiments that may take hours, analysis of the training progress is desirable. Two interviewees explicitly pointed to the need for "early stopping" (P1) of ineffective experiments. P1: *"If I see the loss curve jumping all over the place, or very noisy in comparison to the other loss curves, I don't even need to ever look at that model again, I just thought it out immediately"*.

4.3.3 Informing the next round of exploration

A third general need revolves around the support for making complex decisions. The decision is whether to run a new batch of experiments and, if so, what new set of hyperparameter values to use, given the results of the current batch of experiments. One of the challenges is to monitor and reason about many hyperparameters at the same time. This is particularly evident when training deep learning models. The data science practitioners have to restrict the number of variables to keep in mind due to limited cognitive capacity. When deciding what small subset of the hyperparameter space to explore next, they need to capture the insights from the analysis of the existing experiments (see the first two needs, above). It is worth remarking, about this need, that the decision consists in balancing observations, expectations, and resources: i.e., the performance observed in the current batch of experiments (observations), the desired level of performance given the problem (expectations) and the resources such as time and computation available to the project (resources). While the observations are in the tool, the expectations and resources are mostly implicit knowledge in the head of the data science practitioner - herefrom the need for visual analytics tools that involve the human. As summarized by P4: *"There's never been a point in any project I've ever worked on where hyperparameter tuning was done. It's really just I [judging if] I have seen enough and I'm willing to make a decision about what the hyperparameter should be [to meet expectations]. So its more of a question of timelines and schedules"*.

4.3.4 Project-level memory and communication

The fourth need pertains to memory and communication support at the project level. About memory support, P5 describes the needs to capture what was done to easily recall the analysis trajectory later: *"Now in the report, we have the accuracy (performance), but we do not capture what I changed. Over time we will probably forget what weve done, such as the numbers we have changed. So being able to track them is important. We will be able to go back and see how the model has improved. [A] 'rewinding' [capability]."* Several interviewees (e.g., P1, P6) also mentioned that a project-level summary or report on all experiments should allow filtering, annotating, and comparing experiments: e.g., delete or archive experiment, mark as promising, filter, annotate, tag, select and compare two experiments. For example, P1 reports that his current practice is to compare two experiments at a time, in detail. P6 reports that, at the end of her tuning project, she typically selects the best 2-3 experiments from the list and then runs the models on a new dataset, as a final *"blind validation"* step. Some interviewees suggested that project-level reporting would help collaborate with colleagues and communicate the results to the domain experts who requested the model. In P5's words, *"to be able to communicate to the business sponsors outside our team how well the model is performing, and also, we would use it internally for [guiding future] tuning [...] [and] do a comparison between models as well"*.

5 HyperTuner Prototype and Evaluation

To explore how to leverage visual analytics support in the hyperparameter tuning workflow (Figure 4), we implemented and evaluated HyperTuner, an interactive prototype.

5.1 Implementation and Example Data

HyperTuner is a web-based application implemented on the Django framework [3]. The visualization components are developed with Bokeh [4] and D3.js [1].

Core Concepts in the Prototype. In the prototype, a user launches a training script with multiple hyperparameter settings as a Run, where each setting results in an Experiment.

The user interactions supported around a specific Run correspond to completing a full loop connecting the first four sub-steps of the workflow in Figure 4. Once the experiments are completed, the prototype reports and visualizes the values of the performance metrics obtained from each experiment with respect to the corresponding hyperparameter value settings. The user typically completes multiple runs until satisfied by the results, then selects the best models and reports these as the outcome of the entire model tuning Project.

Prototype Design and Contents. The design of the prototype visualizes the experiments results depending on the data types and values of the hyperparameters and performance metrics, thus is model-agnostic and can be applied to different models (see core concepts above). However, to demonstrate the prototype with real-world use cases, we run a real model tuning project and populated the visualizations with realistic data. Specifically, we built a simple convolutional neural network (CNN) and applied it to predict the MNIST dataset of handwritten digits [2]. As a proof of concept, we streamlined a training set of 60,000 examples and a test set of 10,000 examples.

The prototype can visualize experiment results of different models, substituting the current views with the data types and values of the corresponding hyperparameters and metrics. We made a strategic design decision to use a grid of scatter plot to visualize the training results with minimum manipulation and leave it to data scientists to "perform the art" (P4) of interpretation. It is also a practice to use grids of scatter plots to explore, at an early stage of the analysis, correlations among sets of variables (see visualization in statistical tools such as SPSS and SAS).

Use Cases Based on the data obtained from training the CNN model with 8 experiments, below we describe the prototype implementation through the following use case with two phases:

Sarah is a data scientist and she is building a 10-way classifier with the CNN model to recognize the handwritten digit numbers in the MNIST dataset. She has implemented the model skeleton in a python script with the hyperparameters as tunable variables, and needs to decide what values to use.

> Phase 1: Sarah sets a list of values for each hyperparameter and launches a batch of Experiments in the current Run. After obtaining the training results, she makes sense of these results and decides how to continue the tuning process.

> Phase 2: Sarah stops the tuning progress, cleans up the training records, and saves her progress as a report.

5.2 Phase 1

Set Value to Launch Experiments. Sarah started by experimenting with three hyperparameters: *batch size (number of samples that going to be propagated through the network)*, *dropout rate (the probability at which randomly selected neurons are ignored during training)*, and *number of epochs (one epoch is when an entire dataset is passed forward and backward through the neural network once)*. She sets several candidate values for each of the three hyperparameters (Figure 5 left) and sets the remaining hyperparameters as default values. Notably, by adding an asterisk (*) after the step size she indicates that the step size increases by multiplying the previous value by two rather than adding two each time (e.g. 28,56,128 instead of 28,30,32). Then she selects the metrics she wants to use to measure the model performance (Figure 5 right). In response of her parameter setting actions, the tool automatically generates the command to execute the script via a command-line interface, which she commonly uses to run scripts (see the bottom left field in Figure 5). She can further customize and add more hyperparameters in the final command, and choose to log more performance metrics (the bottom right drop-down menu of Figure 5).

Figure 5: Initial Parameter Setting to Launch Experiments

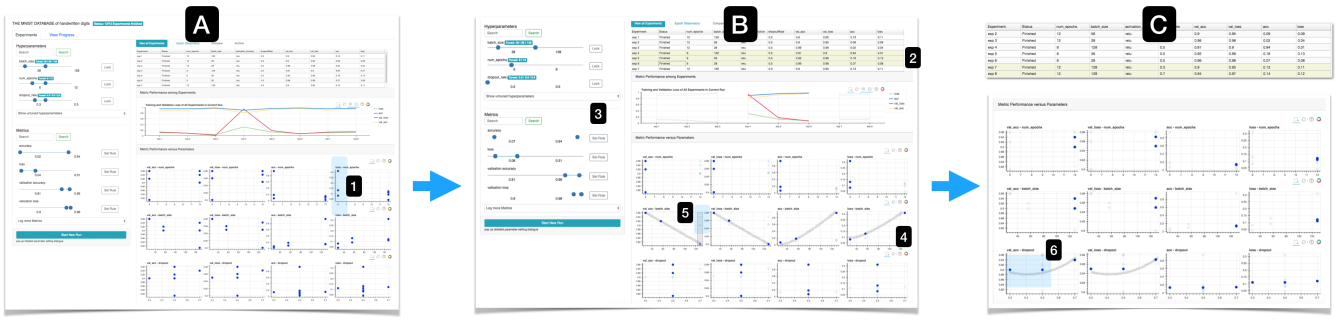

Figure 6: Run dashboard for a batch of experiments in an example interaction flow

Run Dashboard. After the experiments are launched and completed, Sarah reviews the results of all the experiments summarized in the run dashboard (Figure 6 A). By viewing the parameter panel on the left she is reminded of the hyperparameters values she had set when she launched this run plus the metrics she selected to assess performance. For each hyperparameter and metric, she can scan the current value ranges under each slide bar. On the right, she sees both the table and a set of visualizations. The experiment results are summarized in the table at the top: it lists experiment ID, status, hyperparameters tuned, and performance metrics obtained. Under the table, she finds two types of visualizations. The first is an aggregated line chart showing the performance metrics (lines) obtained for each of the eight experiments (x-axis). She can click on the legend to choose which performance metric to view. The second is a grid of 12 scatter plots (three rows, four columns) showing the detailed results for each metric-hyperparameter combination: each row corresponds to one hyperparameter (always shown on the x-axis) and each column corresponds to one performance metric (always shown on the y-axis).

Sarah notices that experiment 4 has worse performance than the others. She suspects that it's because this experiment had a low number of epochs. So by brushing over the top right scatterplot, she selects the experiments with the smaller number of epochs (num_epochs=6, Figure 6 A.1) to see how these experiments performed (number of epochs corresponds to the first row). Since all views in this dashboard are coordinated, the brushing operation results in selecting three experiments across all views, including the table at the top (Figure 6 B.2). It also results into updated sliders in the parameter panel on the left: the lower and upper limit of each range (blue circles) in each slider is automatically re-positioned to reflect the hyperparameter and performance metrics of the experiments selected by the brushing (Figure 6 B.3). At this point, Sarah notices that as the experiments selected have num_epoch=6 and dropout_rate=0.5, the batch_size shows a linear relationship with all the performance metrics (the relationship is highlighted for the reader with gray lines in Figure 6 B.4). Thus she infers that experiments with higher batch_size values might have higher accuracy and lower loss values. Based on this insight, Sarah now selects the experiments with the largest batch_size, batch_size=128 (Figure 6 B.5). Based on this selection she has now identified three experiments (Figure 6 C), and it seems that dropout rates 0.3 and 0.5 are not as good as 0.7. Yet, none of the three experiments has good accuracy, thus she decides to check each experiment in more detail via the experiment dashboard.

Experiment Dashboard. Sarah clicks on the *Epoch Observatory* sub-tab and enters the experiment dashboard, where the left panel and table at the top are persisted from the run dashboard where she was earlier. Here, in the table, she selects one of the three rows (experiments) she is investigating. She replays the training process of the individual experiment (Figure 7 label 1). She repeats this process with the other two experiments. She is investigating how the loss and accuracy curves look like in each experiment, and specifically, each epoch. This will help her find a good trade-off between good final performance and amount of noise (i.e., metric fluctuations) in the training process. In the experiment dashboard, under the table, she analyzes the configuration summaries or metadata of the experiment (label 2), the line charts showing the performance metrics (lines) within epoch (label 3) and across epochs (label 4). In each of these line charts, she clicks on the legend to choose which performance metric to view. On the lower right, she finds a visualization that is specific to the current model and dataset. In this case, she sees a confusion matrix as a heat map. This visualization helps her assess if she can trust the model trained in the current experiment. Specifically, she inspects the cells that show what digits are more frequently misclassified and why by looking at the examples shown, upon cell hovering, under the matrix. For example, she hovers over row 2 and column 6 and finds out that there are 14 data points that are actually digit "2" but classified as digit "6" (see frequency 14 in the matrix, magnified in Figure 7 label 5), and the images at the bottom are examples of those misclassified data points. This gives her a sense of the quality of the model predictions.

Support to Decide the Next Batch of Experiments. After examining the current experiment results in the global and local views

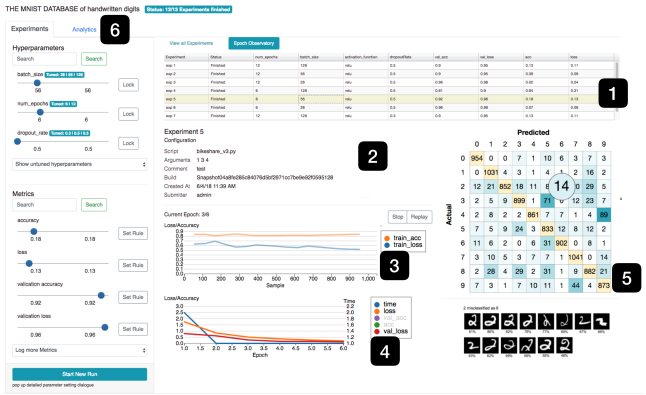

Figure 7: Experiment dashboard for an individual experiment

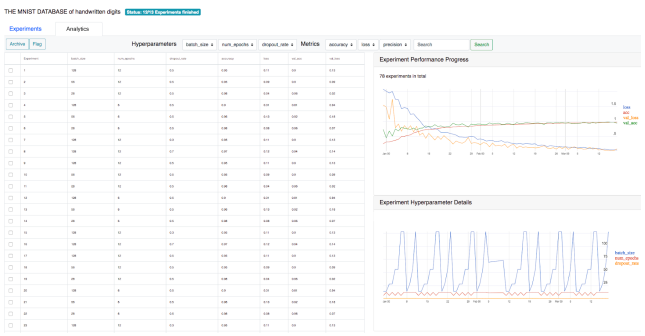

Figure 8: Project Dashboard

back and forth, Sarah decided to run another batch of experiments, where she wants to keep the three tuned hyperparameter values the same as the best experiment of the current run: *num_epochs=12*, *batch_size=128*, *dropout_rate=0.7*. She plans to experiment with a new hyperparameter: *number of layers*, to see if she can use less number of layers to achieve as good a performance. She clicks on "Start new run" on the bottom left and set a new grid of the hyperparameter values in a modal window with a similar interface to Figure 5. As a result, as this new run is complete, the parameter panel in the run dashboard (see Figure 6 B.3 shown earlier) will now show four hyperparameters.

5.3 Phase 2

Project Dashboard. Sarah has tuned the model across five batches of experiments and wants to save her progress so far as she feels satisfied by the results. While still in the Run Dashboard (where she analyzed her last run), she clicks on the "Analytics" tab (Figure 7 label 6) to analyze her progress across all runs in the Project Dashboard (Figure 8) and then create a report. In this dashboard, she first scans the large table on the left with all the experiments she ran and with values of hyperparameter and performance metrics. Then she uses the line chart on the right to review how the performance metrics have improved run after run, historically (Figure 8 upper right) in relation to the tuning of the hyperparameter values used (Figure 8 lower right). By brushing over the upper right chart, she selects all experiments from the first two runs and archives these in a group as the results were poor. Then she cleanses the set of experiments from the remaining three runs by interacting with the table on the left and using the checkboxes in the first column: she selects and archives the bad ones and stars a few good ones that she wants to discuss and annotate with her colleagues. Later on, after meeting her colleagues, she finally includes the best model in the report which she shares with the domain expert who requested the tuned model for a handwriting recognition application.

5.4 Prototype Status

The prototype implementation and evaluation are still in progress. Several components of the prototype, such as features of the project dashboard shown in Figure 8 (e.g., multi-selection, interactions over the charts on the right, and report sharing) are still under construction. The figures (5-8) and use cases in this section are intended to show how the final prototype will support the hyperparameter tuning process. Additional refinements to the prototype implementation are also expected after a final round of evaluation with the same practitioners.

5.5 Preliminary Evaluation

We presented the workflow and the prototype to the same group of six data science practitioners as during the first phase. We used a similar method and each session lasted again an hour. During the session, we used screen sharing to first review the workflow (15 minutes) and then demonstrate each component of the interactive prototype (45 minutes). We used a semi-structured interview method with open-ended questions that requested feedback on the workflow and each prototype component. Each interviewee gave feedback and, when relevant, specified new requirements evoked by the prototype. We recorded and transcribed each session. The qualitative findings extracted by two co-authors from the transcripts are summarized below.

Overall, all interviewees validated that the workflow represents their current hyperparameter tuning process and captures the key sub-steps. The set of visual analytics capabilities supported in the prototype are useful and required. *"I like the feature set, I can imagine myself using it. The handwritten notes stop making sense to me after a long time, and hard to understand for other people. The problem is: you just make up as you go along [reviewing the notes]. This structure will help formalize that." (P1).* In addition, the interviewees found the MNIST dataset and the CNN model used as content for the prototype implementation representative enough of hyperparameters and metrics used for real-world models. The prototype triggered further insights, where the interviewees gave suggestions of how the prototype could be connected or extended to help with work needed after the models are deployed in production (e.g., apply the model to new data).

Run Dashboard. The data science practitioners found the grid of hyperparameter-metric scatter plots and the brushing & linking among all views particularly helpful as it fulfills a need not yet addressed by their tools. *"Its an inherently hard problem how you visualize multiple hyperparameters and performance [metrics]. But giving me the way to slice and dice it definitely helps."* About the line chart showing the performance metrics by experiment, this chart was initially designed as bar charts, then replaced by the current line chart to address scalability concerns by one of the interviewees who usually operate on hundreds of experiments. However, a second interviewee recommended reversing this design decision since connecting the performance with lines might be misleading: *"[The order of the experiments] does not follow a time series relationship ... there's no natural order to them, so I would argue this should be a bar graph, not a time series"*. Another recommended refinement of this chart is to find better solutions for performance metrics that have different scales and units: *"I'm not a big fan of putting accuracy and loss in the same figure."* The visualization (e.g., axes) should interactively adjust based on the metrics selected and show multiple metrics in ways that help to compare a batch of experiments.

Experiment Dashboard. Reviewing the epoch information of each experiment was a recurring practice among the interviewees. Moreover, they found the confusion matrix very helpful. This visualization is specific to the model and the data being used, thus it would be expensive to support across models. Our current prototype design partially addresses this problem by allowing extensions: for models that do not have an equivalent visualization, we leave it to the data science practitioner to plug in their own customized visualizations. However, the interviewees all suggested that it is still worth the efforts to pre-build some visualizations. *I am willing to write extra lines of code to process the data so that it can be visualized this way. (P1).* This suggests the need for pre-building commonly used visualizations as skeletons so that the users can populate with the training results of different models and/or datasets.

Support to Decide the Next Batch of Experiments. Keeping a panel of hyperparameters and performance metrics on the left in both run and experiment dashboards allows the user to keep track of their insights emerged from the analysis of the visualizations (Figure 6 B.3). All interviewees found the "lock" button in the parameter panel useful to record a promising range of values for a hyperparameter. In addition to this basic capability, it might help to have more advanced ways to capture insights during visual analysis. We explored with the interviewees the utility of defining and highlighting patterns in a visualization that may help locate the hyperparameter space worth exploring. For example, the user may build rules based on patterns in the form of threshold values for a metric (e.g., minimum accuracy required) or trend-line slope (e.g., positive or negative hyperparameter-metric relationship) over the hyperparameter-metric scatter plots. However, as the interviewees pointed out *"the challenge is that the patterns are project-specific." (all)* and thus this type of advanced visual analytics support remains a challenge.

The evaluation of the parameter setting window (Figure 5) suggested that a linear step size is usually not enough. Other common cases are logarithmic step sizes or other strategies. Some interviewees suggested that they would also like to manually type in the values in the argument line. Furthermore, they want to receive more feedback as they specify the parameters to be able to predict what would happen if they were to launch the batch of experiments. This refinement of the current design is motivated in particular by cases where the training might take days and require a large amount of computation. *I would like to see if I do choose to use the linear step size, how many experiments and what will the combinations of hyperparameter values look like for each, just to make sure I didn't mess up with my math when setting step sizes. (P1).* Another reason for showing this feedback is to allow the user to decide what to do with the combinations that have been already run. The interviewees pointed to good reasons why they would run the same hyperparameter value combination again: e.g. if the script or the data as changed in the meantime. That being said, the interviewees also wished they had recommendations on the best combinations to run: *"Automatically display what happened when such combination was used would be the most useful thing to do." (P1) "It could be helpful if the system can automatically recommend the ones with the highest potential success." (P4).*

Project Dashboard. The interviewees found it very useful to have a different level of visualizations that aggregates the model performance over time. These are *"really valuable views because it's very easy to spin your wheels and make no progress without realizing it. "(P1).* Some also brought up the need to keep free notes. *"Eight is a small number of experiments, and an interesting hyperparameter search is probably more than this. Then this starts to be challenging to find what you thought of yesterday." (P4).* Archiving and flagging experiments are important, because over time as new data coming in, they want to flag what run to use in production. Such features help decide when to push a model to production and when to bring it out of production. Their common practice is to constantly update the test set to keep an eye on the model performance with the new data. There were also requirements on more visualizations, *"I would also want to see the loss over epoch for multiple models[experiments] on the same graph. (P4)"*.

6 Discussion and Future Work

6.1 Opportunity of learning from user interaction.

An area of future work and discussion central to this workshop is about how visual analytics tools can allow data science practitioners to understand the influence of hyperparameters by capturing patterns on hyperparameter-metrics visualizations and use these patterns to make decisions on the next round of hyperparameter tuning. For instance, there is a known pattern that data science practitioners rely on to decide if the model is overfitting, which indicates that it's time to stop the training. The practice is to monitor both training loss and validation loss as a model is being trained. At the beginning, both the training loss and the validation loss values would decrease. Once the validation loss starts to increase, it means the model starts to overfit the training set of the data. This is a prevalently used pattern that can be predefined in visual analytics tools and be automatically flagged on occurrence in any project.

However, most, if not all, of the other relevant patterns that a data science practitioners could use are project-specific, such as the slope of the training loss or the amount of noise (i.e., value fluctuations or variance) in the learning curve. Often there are no a priori rules, and instead comparisons afterward. *"I don't know what's good until I run what's bad. It's kind of important to understand what caused the noisy curve and how can I remedy noise in the future."* The limited transferability of these patterns across projects due to external factors such as performance expectations and resources not specified in the tool makes it essential to keep human in the loop and learn about ad-hoc user interaction to provide project-specific support. This point was stated emphatically by one of the interviewees (P4): *"hyperparameter tuning is a special part of machine learning, ... an art that doesn't have many libraries or rules of thumb. There are no rules to indicate when to apply what. It's all about trial and errors but because of that it requires experience to know which trials you will never even start doing."*

6.2 Modularized workflow and scalability with higher dimensions.

Another area of future work and discussion is about alternative strategies to search the parameter space and how they could be combined modularly depending on the project needs. The number of hyperparameters of a machine learning model can range from none to more than a hundred. Our prototype assumes an iterative grid search where we picked 3 hyperparameters at a time. This is because according to the data scientists, they usually progressively explore the hyperparameter space rather than all at once, and they would focus on a dozen of most important hyperparameters when the hyperparameter space is high.

The modularization of the workflow we proposed would allow plugging in automated search. For example, an automated random search or more advanced sampling methods like latin hypercubes and low-discrepancy sequences, can be initiated by sampling the candidate values from a user-specified range. Then after the experiments are completed by this automatic module, the results can be used to fit in certain statistical models, depending on the optimization algorithm.

6.3 Limitations and Next Step

Balance model-agnostic and model-specific analysis. One of the challenges in building visual analytics tools for hyperparameter tuning is the balance generalizability and the specific model as well as the data. We demonstrate the prototype with an example

project that uses CNN model to classify MNIST dataset [2]. Most of the visualizations can be customized for training results of different models by specifying the number and data types of the hyperparameters and performance metrics, which do not make assumptions about the type of the model. On the other hand, when deciding the hyperparameter values in the next batch of experiments, more in-depth, model-specific analysis is required. It's worth remarking that the data science practitioners expressed willingness to write additional lines of code to extend the visualizations for more model-specific analysis. For example, the confusion matrix we demonstrated in the individual experiment dashboard is specific to this 10-class classifier example. In the case of a k-means algorithm, it needs to be replaced by an elbow chart by the users.

A special type of hyperparameter: epoch/iterations For single-stage learners like the above-mentioned k-means, or decision trees, the special type of hyperparameter "epochs/iterations" in the CNN use case scenario will not apply anymore. It represents the progress of a particular training process when training is iterative like stochastic gradient descent (where an iteration is usually referred to as an epoch, thus the notion in the prototype). If the user sets the number of epochs to be 10, the performance actually gets tracked for 1 to 10. This would be helpful if the user decides to "interrupt an experiment", which means stopping before it reaches the max epoch since it is noticed that things aren't converging. In some training runs, there's never an epoch since there's so much data you might never process it all. It's also common to take random batches rather than looping through the data set. In that case, users would probably track performance by the batch number.

More rigorous evaluation via case studies. Future evaluation of HyperTuner will include case studies where the data science practitioners actually use the tool to make sense of real-world training results. This will help understand how users capture patterns in visualizations to make sense of the hyperparameter impact and make tuning decisions. In this work, we made an assumption that all the hyperparameter and performance metrics are tracked and available for visualization. One of the interviewees raised the concern regarding how the interface communicates with the scripts. A possibility is to eliminate the code for hyperparameter search and launch the script for running an individual model many times. The person who writes the scripts does not have to update the code every time when there is a need to start a new hyperparameters tuning process.

7 Conclusion

In this paper, we reported the results of our interviews with data science practitioners in industry on the empirical user needs for hyperparameter tuning and presented a workflow to characterized the process with the key steps. We proposed the corresponding visual analytics support to each step and demonstrated early-stage designs via a prototype. Our preliminary evaluation results show that the prototype satisfies the needs of hyperparameter tuning and triggers additional requirements from the users. Future work is required to perfect and extend the design to incorporate more advanced search strategies, as well as more extensive evaluation sessions. We share this early-stage work as an example of how visualization can support hyperparameter tuning and hopefully evoke more discussion and research in this area.

Acknowledgments

The authors wish to thank the data science practitioners for their time and active participation and our collaborators in the UX and CDSW teams at Cloudera for their support.

References

[1] D3.js - Data-Driven Documents.
[2] MNIST handwritten digit database, Yann LeCun, Corinna Cortes and Chris Burges.
[3] The Web framework for perfectionists with deadlines — Django.
[4] Welcome to Bokeh Bokeh 0.13.0 documentation.
[5] S. Afzal, R. Maciejewski, and D. S. Ebert. Visual analytics decision support environment for epidemic modeling and response evaluation. In *Visual Analytics Science and Technology (VAST), 2011 IEEE Conference on*, pp. 191–200. IEEE, 2011.
[6] R. Bardenet, M. Brendel, B. Kégl, and M. Sebag. Collaborative hyperparameter tuning. In *International Conference on Machine Learning*, pp. 199–207, 2013.
[7] J. S. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. Algorithms for hyper-parameter optimization. In *Advances in neural information processing systems*, pp. 2546–2554, 2011.
[8] G. E. Box and K. B. Wilson. On the experimental attainment of optimum conditions. In *Breakthroughs in statistics*, pp. 270–310. Springer, 1992.
[9] M. Claesen and B. De Moor. Hyperparameter search in machine learning. *arXiv preprint arXiv:1502.02127*, 2015.
[10] M. Claesen, J. Simm, D. Popovic, Y. Moreau, and B. De Moor. Easy hyperparameter search using optunity. *arXiv preprint arXiv:1412.1114*, 2014.
[11] A. Coates, A. Ng, and H. Lee. An analysis of single-layer networks in unsupervised feature learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp. 215–223, 2011.
[12] J. Fogarty, D. Tan, A. Kapoor, and S. Winder. Cueflik: interactive concept learning in image search. In *Proceedings of the sigchi conference on human factors in computing systems*, pp. 29–38. ACM, 2008.
[13] R. C. Fong and A. Vedaldi. Interpretable explanations of black boxes by meaningful perturbation. *arXiv preprint arXiv:1704.03296*, 2017.
[14] G. E. Hinton. A practical guide to training restricted boltzmann machines. In *Neural networks: Tricks of the trade*, pp. 599–619. Springer, 2012.
[15] C.-W. Hsu, C.-C. Chang, C.-J. Lin, et al. A practical guide to support vector classification. 2003.
[16] F. Hutter, H. H. Hoos, K. U. Ca, and S. A. Be. ParamILS: An Automatic Algorithm Configuration Framework Kevin Leyton-Brown Thomas StützleStStützle. *Journal of Artificial Intelligence Research*, 36:267–306, 2009.
[17] M. Liu, J. Shi, Z. Li, C. Li, J. Zhu, and S. Liu. Towards better analysis of deep convolutional neural networks. *IEEE transactions on visualization and computer graphics*, 23(1):91–100, 2017. doi: 10.1109/TVCG.2016.2598831
[18] S. Liu, X. Wang, M. Liu, and J. Zhu. Towards better analysis of machine learning models: A visual analytics perspective. *Visual Informatics*, 1(1):48–56, 3 2017. doi: 10.1016/J.VISINF.2017.01.006
[19] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and . Duchesnay. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12(Oct):2825–2830, 2011.
[20] N. Pinto, D. Doukhan, J. J. DiCarlo, and D. D. Cox. A High-Throughput Screening Approach to Discovering Good Forms of Biologically Inspired Visual Representation. *PLoS Computational Biology*, 5(11):e1000579, 11 2009. doi: 10.1371/journal.pcbi.1000579
[21] K. Potter, A. Wilson, P.-T. Bremer, D. Williams, C. Doutriaux, V. Pas, and C. R. Johnson. Ensemble-Vis: A Framework for the Statistical Visualization of Ensemble Data. In *2009 IEEE International Conference on Data Mining Workshops*, pp. 233–240. IEEE, 12 2009. doi: 10.1109/ICDMW.2009.55
[22] A. J. Pretorius, M. A. P. Bray, A. E. Carpenter, and R. A. Ruddle. Visualization of parameter space for image analysis. *IEEE Transactions on Visualization and Computer Graphics*, 2011. doi: 10.1109/TVCG.2011.253
[23] M. T. Ribeiro, S. Singh, and C. Guestrin. Why should i trust you?: Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 1135–1144. ACM, 2016.
[24] D. Sacha, M. Sedlmair, L. Zhang, J. A. Lee, D. Weiskopf, S. North, and D. Keim. Human-Centered Machine Learning Through Interactive Visualization: Review and Open Challenges. ESANN, 2016.

[25] D. Sacha, H. Senaratne, B. C. Kwon, G. Ellis, and D. A. Keim. The Role of Uncertainty, Awareness, and Trust in Visual Analytics. *IEEE Transactions on Visualization and Computer Graphics*, 22(1):240–249, 1 2016. doi: 10.1109/TVCG.2015.2467591

[26] M. Sedlmair, C. Heinzl, S. Bruckner, H. Piringer, and T. Moller. Visual parameter space analysis: A conceptual framework. *IEEE Transactions on Visualization and Computer Graphics*, 2014. doi: 10.1109/TVCG.2014.2346321

[27] J. Tsay, T. Mummert, N. Bobroff, A. Braz, P. Westerink, and M. Hirzel. Runway: machine learning model experiment management tool, 2018.

[28] F.-Y. Tzeng and K.-L. Ma. Opening the black box-data driven visualization of neural networks. In *Visualization, 2005. VIS 05. IEEE*, pp. 383–390. IEEE, 2005. doi: 10.1109/VISUAL.2005.1532820

[29] J. Wang, L. Gou, H. Yang, and H.-W. Shen. Ganviz: A visual analytics approach to understand the adversarial game. *IEEE transactions on visualization and computer graphics*, 24(6):1905–1917, 2018.

[30] D. Yogatama and G. Mann. Efficient transfer learning method for automatic hyperparameter tuning. In *Artificial Intelligence and Statistics*, pp. 1077–1085, 2014.