SINGA-Easy: An Easy-to-Use Framework for MultiModal Analysis

Naili Xing[1], Sai Ho Yeung[1], Cheng-Hao Cai[1,4], Teck Khim Ng[1], Wei Wang[1], Kaiyuan Yang[1], Nan Yang[1], Meihui Zhang[2], Gang Chen[3], Beng Chin Ooi[1]
[1]National University of Singapore, Singapore [2]Beijing Institute of Technology, China
[3]Zhejiang University, China [4]National University of Singapore (Suzhou) Research Institute, China
{xingnl,yeungsh,caich,ngtk,yangky,yangn,ooibc}@comp.nus.edu.sg
wangwei.cs@gmail.com meihui_zhang@bit.edu.cn cg@zju.edu.cn

ABSTRACT

Deep learning has achieved great success in a wide spectrum of multimedia applications such as image classification, natural language processing and multimodal data analysis. Recent years have seen the development of many deep learning frameworks that provide a high-level programming interface for users to design models, conduct training and deploy inference. However, it remains challenging to build an efficient end-to-end multimedia application with most existing frameworks. Specifically, in terms of usability, it is demanding for non-experts to implement deep learning models, obtain the right settings for the entire machine learning pipeline, manage models and datasets, and exploit external data sources all together. Further, in terms of adaptability, elastic computation solutions are much needed as the actual serving workload fluctuates constantly, and scaling the hardware resources to handle the fluctuating workload is typically infeasible. To address these challenges, we introduce SINGA-Easy, a new deep learning framework that provides distributed hyper-parameter tuning at the training stage, dynamic computational cost control at the inference stage, and intuitive user interactions with multimedia contents facilitated by model explanation. Our experiments on the training and deployment of multi-modality data analysis applications show that the framework is both usable and adaptable to dynamic inference loads. We implement SINGA-Easy on top of Apache SINGA and demonstrate our system with the entire machine learning life cycle.

CCS CONCEPTS

• **Information systems → Multimedia information systems**; • **Computing methodologies → Machine learning**; • **Human-centered computing → Systems and tools for interaction design**.

KEYWORDS

deep learning, data analytics, multimedia application, distributed training, dynamic inference

ACM Reference Format:
Naili Xing[1], Sai Ho Yeung[1], Cheng-Hao Cai[1,4], Teck Khim Ng[1], Wei Wang[1], Kaiyuan Yang[1],, Nan Yang[1], Meihui Zhang[2], Gang Chen[3], Beng Chin Ooi[1]. 2021. SINGA-Easy: An Easy-to-Use Framework for MultiModal Analysis. In *Proceedings of the 29th ACM International Conference on Multimedia (MM '21), October 20–24, 2021, Virtual Event, China.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3474085.3475176

MM '21, October 20–24, 2021, Virtual Event, China

ACM ISBN 978-1-4503-8651-7/21/10.
https://doi.org/10.1145/3474085.3475176

1 INTRODUCTION

Deep learning has been successfully adopted in a variety of multimedia applications such as image classification, speech recognition and news recommendation. Driven by the increasing demand of real-world multimedia applications and an unprecedented growth of big data, many Deep Learning (DL) techniques and systems [6, 44, 45] have been developed to facilitate the development of AI applications. Although human-level performance has been achieved in areas like computer vision [47], natural language processing [13] and speech processing [46], the mass adoption of AI applications remains elusive due to two major challenges.

The first challenge is usability [37]. Many AutoML frameworks have been developed to improve usability. These include Auto-WEKA [22], H20 AutoML [24], Auto-Sklearn [11], Auto-Pytorch [49], and Auto-Keras [20]. Among these frameworks, some provide functionalities for hyper-parameter tuning that can work with large datasets, and some support good user interaction and experience. However, most of them do not take into account both.

Specifically, for hyper-parameters tuning, most DL model training processes focus on searching for the best hyper-parameter configuration using a set of stochastic gradient descent based optimization algorithms [3, 4, 21]. Many AutoML systems [25, 28] use Bayesian Optimization (BO) [33, 40] to tune the hyper-parameters automatically. This is often time consuming as they need to evaluate many different combinations of hyper-parameters to obtain the best configuration. To accelerate the searching process, AI-as-a-Service platforms such as Rafiki [43] tune the hyper-parameters of the SGD algorithms in a distributed manner, but it pays little attention to the model (or architecture) related hyper-parameters.

In terms of user interaction and experience, DL systems typically hide implementation details and appear like a black-box to users. To provide better interaction and user control, an easy-to-use APIs for managing the ML job in a finer-grained manner is required. In addition, a good model explanation solution [34, 36, 38] is also much needed in real-world applications, especially for those high-stakes applications. For example, in the X-ray based medical diagnosis [12, 30, 42], making a wrong decision may lead to catastrophic consequences, and meanwhile, providing explainable AI solutions [14, 18, 41] can also engender user trust. Auto-WEKA and H20 AutoML provide graphical user interfaces for datasets, models and task management. In addition, H20 AutoML provides a number of model explanation functions based on variable importance and dependency. However, most of these existing systems do not provide full support for automatic hyper-parameter tuning, good user interaction and model explanation.

The second challenge is adaptability – inference services have to support elastic computation control in real-time, as it is not practical and often infeasible to scale the computational resources of the system to handling peak workloads. In practice, the users may want to obtain more prediction results within a stipulated time and meanwhile can tolerate a slight decrease in accuracy. A conventional static model takes a fixed amount of computation and thus can not trade off accuracy and efficiency dynamically, which therefore is unable to handle the fluctuating workload. Clipper [8] focuses on the prediction serving by introducing a new framework and explores several optimization techniques such as caching and model selection to improve latency and accuracy. However, it simply drops the instances if they cannot be processed within the time limit in the presence of high workloads. Model-Switching [48] - an online scheduler on top of Clipper, can select and switch to a different serving model based on the budget dynamically, which can achieve higher *effective accuracy*. Nevertheless, multiple models need to be trained beforehand and loaded to memory to support runtime model selection, which incurs additional overheads. Notably, the ability to efficiently and effectively adapting the serving model size to the current workload and computational resources available is still lacking in most systems.

To address these two challenges, namely the usability and adaptability, an easy-to-use deep learning framework supporting automatic hyper-parameter selection, distributed training, dataset and model management, model explanation, and elastic computation control is required. We design and implement such a system on top of Apache SINGA [29]. The main contributions of this work are summarized as follows:

- We present an end-to-end open-sourced DL framework called SINGA-Easy, which is developed to facilitate the adoption of DL algorithms and inference services by domain-specific multimedia application users. SINGA-Easy can automatically tune training jobs for pre-built complex models and adapt the model size when facing high inference workloads. It also provides an intuitive APIs to manage the whole DL/ML life cycle and retrieve the inference result from various perspectives.
- At the training stage, we propose a new auto-tuning framework combining both a distributed hyper-parameter tuning policy and an adaptive regularization method, which reduce the effort required for training an efficient and accurate model. We also integrate to our system a novel technique to adapt the serving model size dynamically.
- At the inference stage, we focus on the evaluation metric *effective accuracy* and propose a new scheduling algorithm to adapt the model size to the current workload in real-time for achieving higher *effective accuracy*, and meanwhile satisfy user-defined response time requirement under the available computational resources. The algorithm also reduces manual effort in the deployment, scaling and workload balancing of the service.

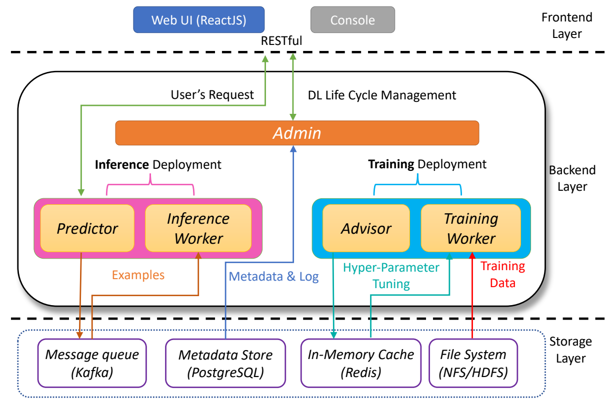


Figure 1: SINGA-Easy system architecture overview.

- To facilitate the adoption of multimedia applications, we integrate commonly used algorithms and provide an easy-to-use APIs. We also provide an option for users to evaluate model performance from the model explanation perspective provided by LIME [36] and Grad-CAM [38].
- We demonstrate the usability and adaptability of our SINGA-Easy by conducting experiments on various real-world datasets and showcasing several multimedia applications.

The remainder of the paper is structured as follows. Section 2 introduces the system architecture and the dataflow between the system components. Section 3 introduces the dynamic model serving framework. Section 4 presents system usability. Section 5 presents the experimental study. We review the related work in Section 6 and conclude the paper in Section 7.

2 SYSTEM ARCHITECTURE

This section introduces the system architecture of SINGA-Easy. The software stack is illustrated in Figure 1 and Figure 2 respectively. The system consists of the *frontend layer*, the *backend layer*, and the *storage layer*. Specifically, the frontend layer provides different HTTP APIs to manage both data and tasks. Users can interact with the framework via either *Python SDK-Client* or *Web UI*. In the following subsections, we will introduce the other layers in detail.

2.1 Backend Layer

SINGA-Easy is built on top of the base architecture of Apache SINGA. The backend of the overall system comprises five essential components: *Admin*, *Training Worker*, *Advisor*, *Predictor* and *Inference Worker*.

Admin is the core component of the system's control plane, which exposes HTTP APIs for users to manage the whole ML lifecycle. Upon receiving requests from users via RESTful APIs, it deploys a number of workers for model training and serving, and stores information of the user-defined tasks into a *Metadata Store*. When a worker is deployed, the worker pulls the information of the task from the *Metadata Store* and starts the task.

Training Worker trains models by conducting trials proposed by a corresponding *Advisor Worker*. The computational kernel of *Training Worker* supports various DL libraries in addition to Apache SINGA e.g., PyTorch [31], and TensorFlow [1]. Figure 2

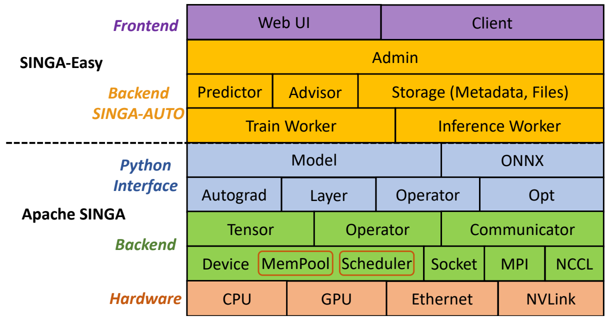


Figure 2: SINGA-Easy software stack.

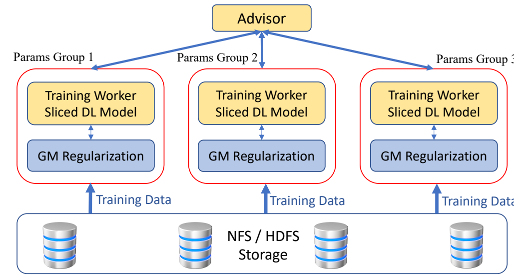


Figure 3: Training Worker: distributed model training.

shows the stack diagram with Apache SINGA as the DL framework, where the upper layers are constructed based on the lower layers. For example, the component *model* is defined using *Layer*, *Autograd*, *Opt* and *Operator*, etc. Each of them is built on top of the basic data structures of Apache SINGA such as *Tensor* and *Communicator*. We introduce the technical details of the training part in Section 2.4.

Advisor performs hyper-parameter tuning by conducting multiple trials on a training job. In each trial, it proposes the training configuration, i.e., knobs of the model and training algorithm, to be used by *Training Worker*. For the implementation of the Advisor, we adopt the Bayesian Optimization of Scikit-Optimize toolbox[1].

Predictor is designed for ensemble modelling, which stands between users and *Inference Workers*. *Predictor* receives requests, e.g., one or many images to be classified, from the users, then forwards the requests to a number of *Inference Workers* and collects the prediction results. *Inference Worker* manages trained models for the inference jobs, which receives the request forwarded by the *Predictor* and performs prediction. The technical details of inference are discussed in Section 3.

2.2 Storage Layer

The storage layer contains the following components for caching and storage of data:

Metadata Store is a centralized and persistent database used to store the metadata of the whole system such as user metadata, job metadata, worker metadata and model templates. We adopt PostgreSQL[2] in our system.

In-Memory Cache is an in-memory data structure store used for fast asynchronous communication between *Training Workers* and *Advisor* at the training stage. Redis[3] is used as the in-memory data store.

Message Queue is a file-based data store used for supporting asynchronous communication between *Inference Workers* and *Predictors* for Inference Jobs. Apache Kafka[4] is used in our system.

2.3 Workflow

SINGA-Easy allows users to manage the whole ML life cycle and retrieve the prediction results from inference services. The user firstly uploads the model, dataset, or annotation file to *Admin*, which will be stored into a distributed storage (NFS). The metadata will be stored into *Metadata Store*. When use send requests to start training, *Admin* launches one *Advisor* and multiple *Training Workers*. *Advisor* stores training configurations into *In-Memory Cache*, and *Training Workers* will conduct training accordingly.

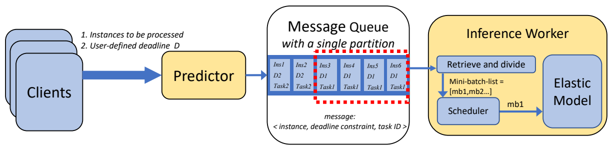
In each iteration, *Training Workers* report the training accuracy to *In-Memory Cache*, which is then used by *Advisor* to generate new configurations for the next training iteration. After completing a training job, one *Predictor* and multiple *Inference Workers* will be created. The user can retrieve the *Predictor* inference service's URL from *Admin* to use the model inference services. All these components communicate with each other via a message queue.

2.4 Elastic Inference

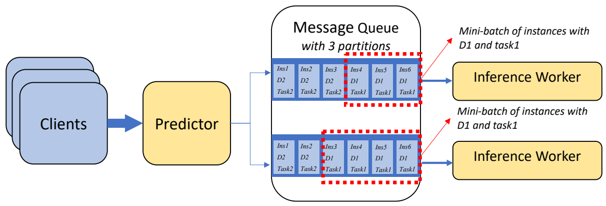
Model slicing [5] is a general technique to enable deep learning models to support elastic computation. Specifically, each layer of the model is divided into equal-sized contiguous computational groups. During both training and inference, there is a single parameter slice rate r that dynamically controls the fraction of groups involved in the computation for all the layers in the model, namely the model width. In particular, these groups are trained dynamically to build up representations residually. The first group learns the base representation. Each subsequent group learns on top of all its preceding groups. As a result, during inference, we can support accuracy-efficiency trade-offs by dynamically slicing a subnet of a certain width, where only the parameters of the activated groups are involved in computation. Theoretically, the number of parameters and computation measured in FLOPs are both roughly quadratic to the slice rate r [5], e.g., a slice rate of 0.5 can achieve up to four times speedup. Therefore, we can support elastic inference by introducing the model slicing technique to the training stage. Specifically, we can train the model with multiple slice rates beforehand. At the inference stage, the model can be switched to different sub-models adaptively.

The overall training process is illustrated in Figure 3. We train the model with model slicing to render the ability of elastic computation. For efficiency, the system trains the sub-models in a distributed manner by training these sub-model instances of different slice rates in a pool of workers. After a few training iterations, all workers merge their local copy of weights and update them globally. SINGA-Easy reuses the distributed hyper-parameter tuning component of Apache SINGA and adds an adaptive Gaussian Mixture (GM) regularization technique [27] to further improve the prediction performance of the model.

[1]Scikit-Optimize: https://scikit-optimize.github.io/stable
[2]PostgreSQL: https://www.postgresql.org/
[3]Redis: https://redis.io
[4]kafka: https://kafka.apache.org



(a) Single model serving



(b) Multiple model serving

Figure 4: Inference stage: each message in the message queue contains one instance and the global deadline constraint for this prediction task.

3 DYNAMIC MODEL SERVING

To support dynamic model serving, we further propose a scheduling algorithm based on the elastic inference enabled via model slicing. At the inference stage, the user can send multiple requests, each of which corresponds to one prediction task. A prediction task may contains multiple instances to be processed by the serving model. As shown in Figure 4(a), each *Client* sends instances and a global deadline constraint D to *Predictor*. The *Predictor* and *Inference Workers* will then work cooperatively to produce the inference result. The *Predictor* wrap each instance to be a message in the form of 3-tuple $< Instance, D, TaskID >$, it then pushes all messages to a message queue as a producer. The *Inference Workers* read message of the same prediction task from the queue as consumers, divide the instances into mini-batches, adjust the model size according to the scheduling algorithm and then conduct the inference. The mini-batch is the minimum scheduling unit, which can contain one or more instances, e.g., one video for the video objection detection task or several texts for the text classification task. To fully utilize the GPU parallel computing capability, the size of the mini-batch is typically set to the largest possible. The algorithm is outlined in Algorithm 1. For ease of reference, we summarize all the variables used in subsequent sections in Table 1.

To evaluate the effectiveness of the proposed dynamic model serving, we adopt the *effective accuracy* [48] as the evaluation metric, which is fraction of correctly processed instances returned before a predefined deadline. Specifically, denoting the prediction accuracy of a given model as p and the fraction of instances that can be processed by the model before the deadline D as f^T, *effective accuracy* p^{eff} can then be computed by $p^{eff} = p * f^T$. Briefly, *effective accuracy* takes into account both the accuracy and efficiency of the serving model, as p represents the prediction accuracy. Given the deadline, f^T is determined by the efficiency of the model inference. Notably, f^T can also be seen as the throughput within the given time frame. The goal of the scheduling algorithm is formally defined as follows.

Given the number of instances to be processed N and a global deadline constraint D, the scheduling algorithm is to maximize the effective accuracy under the available computational resources in the model serving system.

Table 1: Summary of variables.

Symbol	Meaning
m_i	The i-th sub-model
r_i	Slice rate of m_i
p_i	Accuracy of m_i
t_i	Time for m_i to process a mini-batch
p_i^{eff}	*Effective accuracy* of m_i
K	Number of trained sub-models
n_i	Number of mini-batches assigned to m_i
$\widehat{W_i}$	The maximum workload* that m_i can handle
N	Number of instances in one prediction task
S_{mb}	Number of instances in a mini-batch, which is fixed on each prediction task.
N_{mb}	Number of mini-batch, $N_{mb} = N \ / \ S_{mb}$
D	User-defined deadline to process N instances
W_{exp}	Expected workload of the system, $W_{exp} = N \ / \ D$
T_i	Time spent by m_i to process N instances
T_{fast}	T_i of the fastest sub-model
T_{slow}	T_i of the slowest sub-model

*Workload: the number of instances to be processed per second.

We develop two scheduling algorithms in two respective scenarios, where the available computational resources can support either only one model in Section 3.1 or multiple models in Section 3.2. In what follows, we denote that sub-model m_i indexed by the slice rate r_i has a prediction accuracy p_i and takes t_i time on average to process a mini-batch of a fixed number of instances. Typically, the full model (i.e., $r_i = 1.0$) has the highest accuracy while the lowest efficiency. For both scenarios, there are K sub-models trained beforehand. The N instances are divided into N_{mb} mini-batches (each with $\frac{N}{N_{mb}}$ instances) by *Inference Worker*. The sub-model m_i takes T_i to process all N_{mb} mini-batches. Formally, the scheduling algorithms are to determine the best scheduling policy denoted as $[n_1 ... n_m]$, where n_i is the number of mini-batches assigned to sub-model m_i.

3.1 Single Model Serving

In the first scenario where only one single model can be deployed in the system, Algorithm 2 is adopted to dynamically adapt the model size to meet the deadline requirement and to obtain the best *effective accuracy*. The situations are summarized as follows:

- If $D \le T_{fast}$ (see Figure 5 where $D \le 4$), dropping mini-batch will be unavoidable. In this case, the scheduler will schedule all mini-batches to the fastest sub-model to minimize the drop rate.
- If $D \ge T_{slow}$ (see Figure 5 where $D \ge 24$), the scheduler will schedule all mini-batches to the slowest but most accurate sub-model.
- If $T_{fast} < D < T_{slow}$ (see Figure 5 where $4 < D < 24$), more than one sub-model is needed to achieve the best *effective accuracy*.

For a single sub-model m_i, it's *effective accuracy* is defined as:

$$p_i^{eff} = \begin{cases} p_i, & W_{exp} \le \widehat{W_i} \\ (\widehat{W_i} \ / \ W_{exp}) * p_i, & W_{exp} > \widehat{W_i} \end{cases} \tag{1}$$

Algorithm 1 Model Serving *Predictor* and *Inference Worker*

Input: $\mu = [m_1, \ldots, m_K]$, $\rho = [p_1, \ldots, p_K]$, S_{mb}, user's requests
Output: Prediction results

function PREDICTOR
  while True **do**
    Receive user's request with instances, D
    generate task ID for this prediction task
    message: $< Instance, D, TaskID > \longleftarrow$ each instance
    Send the messages to the message queue
  end while
end function
function INFERENCE WORKER(μ, ρ, S_{mb})
  Get $\tau = [t_1, \ldots, t_K]$ according to S_{mb}
  while True **do**
    Retrieve N messages with the same ID from the queue
    Retrieve instances and D from message tuples
    Divide the instances into N_{mb} mini-batches of size S_{mb}
    Store the mini-batches as a list γ
    $\eta, p^{eff}(\eta) \longleftarrow$ `Scheduler`(ρ, τ, D, N_{mb})
    for n_i in η **do**
      for j in $[1, \ldots, n_i]$ **do**
        $R \longleftarrow \mu[i]$.`prediction`$(\gamma.pop())$
        Send R as prediction results to the user
      end for
    end for
  end while
end function

When the expected workload W_{exp} is higher than the maximal workload that a single sub-model can handle, i.e., $\widehat{W_i}$, the *effective accuracy* decreases since the sub-model cannot process all instances before the deadline. In this case, the *effective accuracy* can be improved using multiple sub-models. The optimization objective now becomes a combinatorial optimization problem that maximizes the follows:

$$p^{eff}([n_1, \ldots, n_K]) = \sum_{i=1}^{K} \left(\frac{n_i}{N_{mb}} * p_i\right) \tag{2}$$

Moreover, n_i is the number of mini-batches assigned to the i-th sub-model, and at most N_{mb} mini-batches can be scheduled, we thus have the following bound functions:

$$n_i \geq 0 \ (i = 1, 2 \ldots K) \tag{3}$$

$$n_i \in \mathbb{Z} \ (i = 1, 2 \ldots K) \tag{4}$$

where $\mathbb{Z}$ is the set of integers. We also have:

$$\sum_{i=1}^{K} n_i \leq N_{mb} \tag{5}$$

Additionally, the total time to process all of the scheduled mini-batches is limited by D:

$$\sum_{i=1}^{K} (n_i * t_i) \leq D \tag{6}$$

The maximization of the objective function Eq. (2) with the constraints of Eq. (3), (4), (4), (5) and (6) can be formulated as a

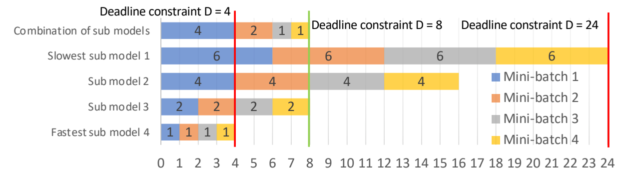


Figure 5: The total Time required to process all four mini-batches using different sub-models.

Integer Linear Programming (ILP) problem, which can be solved by either the classical linear programming-based *Branch-and-Bound (B&B)* method or Dynamic Programming (DP). Although DP can find the optimal solution in polynomial time, the solution may not be precise as D needs to be discretized, which forms a limited number of sub-problems. Thus, the B&B method is used prior to DP, which is shown in Algorithm 2. Specifically, the B&B method takes the following steps: <s1> An initial linear programming problem X_0 is constructed by grouping Eq. (2), (5), (3) and (6). Then X_0 is pushed to the problem queue ϕ. <s2> Retrieve a linear programming problem X from ϕ and get its optimal solution β by applying linear programming method. <s3> If all elements in β are integers, then β satisfies Eq. (4) and will be a feasible solution of the ILP problem. If β leads to a higher p^{eff}, then β will be used to update η. <s4> If β contains non-integers, eg. $\beta[i]$ is a float number, then two new sub-problems are generated by merging the problem X and two respective new constraints, namely $n_i \geq int(\beta[i]) + 1$ and $n_i \leq int(\beta[i])$, which are then pushed into ϕ. Steps <s2>, <s3> and <s4> are iterated until ϕ is empty. Finally, the optimal η can be obtained. If the B&B method fails to find the optimal η, the ILP problem will be approximated as a classical 2-dimensional unbounded knapsack problem and solved by DP.

Our experiments further confirm that the optimal solution of Eq. (2) is also the best scheduling policy that achieves the highest *effective accuracy*. For example, as shown in Figure 5, when the user sets $D = 8$, sub-model 1 can only serve one mini-batch within the deadline. Sub-models 3 and 4 can meet the deadline but with relatively low accuracy. The best scheduling policy is a combination of sub-models 2, 3 and 4 that will provide relatively high accuracy and achieve a zero-drop rate. Specifically, mini-batch 1 is assigned to sub-model 2. The model is then switched to sub-model 3 to process mini-batch 2 after processing mini-batch 1. Finally, mini-batch 3 and 4 are processed using sub-model 4.

To further save the time spent on running the scheduling algorithm, we precompute and store combinations of D and N_{mb} in *In-Memory Cache* to accelerate decision making.

3.2 Multiple Model Serving

In the second scenario where multiple models can be loaded to the system, as shown in Figure 4(b), the *Producer* will partition instances to different queues. Each queue is served by a dedicated *Inference Worker*. The models in other *Inference Workers* are replicated from the first *Inference Worker*. The model in each *Inference Worker* is able to switch between sub-models. The global *effective accuracy* is defined as the average of *effective accuracy* of each *Inference Worker*. Suppose there are b *Inference Workers*, the *global effective accuracy*$=\frac{1}{b}\sum_{j=1}^{b} p_j^{eff}$. The global best *effective accuracy* is equal to the average of the local best *effective accuracy*. To get

Algorithm 2 Scheduling Algorithm

Input: $\rho = [p_1, \ldots, p_K]$, $\tau = [t_1, \ldots, t_K]$, D, N_{mb}
Output: $\eta = [n_1, \ldots, n_K]$, theoretical $p^{eff}(\eta)$

function Scheduler(ρ, τ, D, N_{mb})
  $\eta \leftarrow [0, \ldots, 0]$
  if $D \leq N_{mb} \times \texttt{Min}(\tau)$ **then** $\eta[\texttt{Argmin}(\tau)] \leftarrow N_{mb}$
  end if
  if $D \geq N_{mb} \times \texttt{Max}(\tau)$ **then** $\eta[\texttt{Argmax}(\tau)] \leftarrow N_{mb}$
  end if
  if $N_{mb} \times \texttt{Min}(\tau) < D < N_{mb} \times \texttt{Max}(\tau)$ **then**
    $\phi \leftarrow \texttt{Queue}([\{\text{Eq.}(2), (3), (5), (6)\}])$
    while $\phi \neq []$ **do**
      $X \leftarrow \phi.\texttt{Pop}$
      $\beta \leftarrow \texttt{Linear\_Programming}(X)$
      if $p^{eff}(\beta) > p^{eff}(\eta)$ **then**
        if $\forall i.\ \beta[i] \in \mathbb{Z}$ **then** $\eta \leftarrow \beta$
        else select i such that $\beta[i] \notin \mathbb{Z}$
          $\phi.\texttt{Push}(X \cup \{n_i \geq \texttt{Int}(\beta[i]) + 1\})$
          $\phi.\texttt{Push}(X \cup \{n_i \leq \texttt{Int}(\beta[i])\})$
        end if
      end if
    end while
    // If B&B fails to find a solution
    if $\eta = [0, \ldots, 0]$ **then**
      $\eta \leftarrow \texttt{2D\_Unbounded\_Knapsack\_DP}(\rho, \tau, D, N_{mb})$
    end if
  end if
  return $\eta, p^{eff}(\eta)$
end function

```
client.create_dataset(name='X_ray', task="IMAGE_CLASSIFICATION",
                      dataset_path="./X_rayTrainDataset")
client.create_model(name='RestNet', task='IMAGE_CLASSIFICATION',
                    model_file_path='./RestNet.py',
                    model_class='RestNet',
                    dependencies={"torch": "1.0.1", "torch_vision": "0.2.2"})
client.create_train_job(app='X-rayImagesClassifications',
                        task='IMAGE_CLASSIFICATION',
                        train_dataset_id=1,
                        val_dataset_id=2,
                        budget={})
model_serve = client.create_inference_job(app='X-rayImagesClassifications',
                                          app_version=1,
                                          budget={"GPU": 1})
prediction_result = requests.post(url=model_serve.URL,
                                  data={"image": ["x_ray_image1.jpg",
                                                  "x_ray_image2.jpg"],
                                        "deadline": 5})
prediction_result["model_explanation"].show()
```

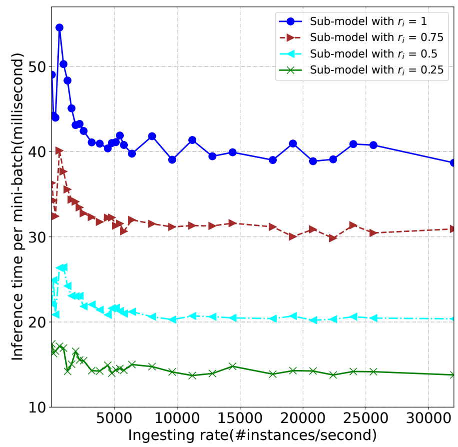
Figure 6: SINGA-Easy APIs for supporting X-ray image classification.

the best local *effective accuracy*, each *Inference Worker* runs Algorithm 2 separately under the original deadline constraint D and the number of mini-batches N_{mb} in the corresponding queue.

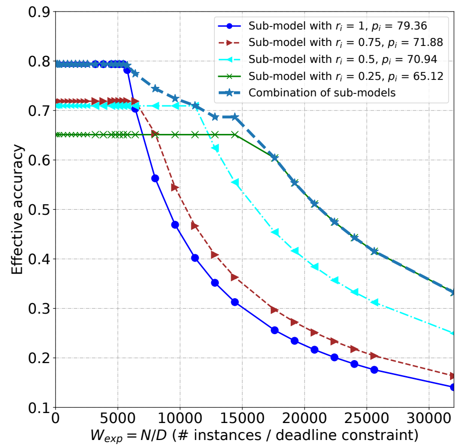
4 SYSTEM USABILITY

To improve the usability of ML and DL models in multimedia applications such as medical image classification, food recognition, dietary management, question answering, and speech classification, SINGA-Easy provides built-in models using third-party libraries based on PyTorch [31], TensorFlow [1] and Scikit-learn [32]. Table 2 lists representative models for six common multimedia tasks.

Figure 6 shows an example on the use of SINGA-Easy's APIs to quickly develop applications with the supported models. To improve the reliability of inference, we have also implemented two model explanation techniques, i.e., Grad-CAM [38] and LIME [36].



(a) Inference time of different sub-models.



(b) Illustration of the impact of the scheduling algorithm on *Effective Accuracy* (ResNet-50 trained with model slicing on NIH Chest X-rays).

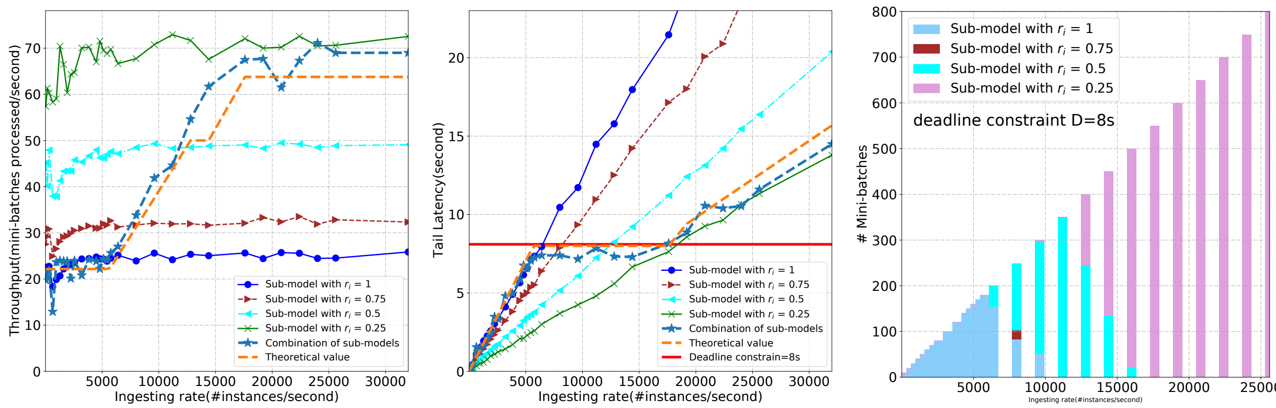
Figure 7: Inference time and *effective accuracy*.

5 EXPERIMENTAL EVALUATION

To evaluate the usability and adaptability of SINGA-Easy, we conduct experiments on different multi-media datasets using various DL network architectures. SINGA-Easy is deployed on clusters equipped with GTX1080Ti GPUs and two models of CPUs. The CPU model deployed in *Admin* node is Intel Xeon CPU E5-2620 v4 @ 2.10GHz. The CPU model deployed in the other nodes is Intel Xeon CPU E5-1650 v4 @ 3.60GHz. All nodes are connected via Ethernet at 1 Gbit/s. We run all services in Docker and use Kubernetes 1.6 as the cluster manager.

5.1 Experimental Setup

In this section, we introduce the training details, namely the datasets adopted for the evaluation and the training results.



(a) Throughput (b) Tail latency (c) Ingesting rate on scheduled sub-models.

Figure 8: Adaptability Experiments in SINGA-Easy

Table 2: Models for multimedia applications.

Model	Scenario
VGG [39], ResNet [16]	Image Classification
Mask RCNN [15]	Object Detection
Deep Speech [2]	Speech Recognition
BERT [10]	Question and Answering
Random Forest [17]	Tabular Classification
XGBoost [7]	Tabular Regression

5.1.1 Datasets. We evaluate the training and inference efficiency of SINGA-Easy using model ResNet-50 on three image classification datasets, namely CIFAR [23], ILSVRC 2012 [9] and NIH Chest X-rays dataset[5]. To demonstrate the general support of SINGA-Easy for different applications, we train Inception-ResNet-v2 and yoloV3 [35] on five food datasets for food image classification and detection and visualize the results. We use 50,000 training images and 10,000 test images from the CIFAR dataset. Each CIFAR image is resized to 32×32. We use 1.2 million training images and 50,000 test images drawn from 1,000 classes in the ILSVRC 2012 dataset. And resize them to 244×244. We also use 5,234 training images and 634 test images from the NIH Chest X-rays dataset. Each image of the X-rays dataset is classified as "healthy" or "unhealthy" and is normalized from 1600×1125 to 244×244. We also use five Singapore food datasets. The number of classes in each dataset is 55, 101, 172, 231 and 256, respectively. Each class contains 300 to 500 images of size 624×1024.

5.1.2 Training Results. ResNet-50 is trained on each dataset with SGD. Specifically, we train 100/100/300 epochs on CIFAR-10/ILSVRC 2012/NIH X-arays with a batch size of 128/64/64, respectively. We summarize the statistics of the trained models in Table 3.

5.2 Dynamic Model Serving Evaluation

The adaptability of SINGA-Easy is evaluated using ResNet-50 trained on dataset NIH Chest X-rays. In the following experiments, we set the mini-batch size S_{mb} to 32 for the evaluation.

[5]NIH Chest X-rays: www.kaggle.com/nih-chest-xrays/data

Table 3: Accuracy and inference time of sliced ResNet-50.

Slice Rate	CIFAR-10		ImageNet12		X-Ray	
	Acc.	T_{ave}	**Acc.**	T_{ave}	**Acc.**	T_{ave}
1	91.13	12.48	75.09	49.40	79.37	45.12
0.75	88.41	9.92	73.74	38.08	71.88	34.56
0.5	85.19	6.41	71.09	22.95	70.94	22.72
0.25	79.71	3.24	63.91	17.82	65.12	15.68

*T_{ave}: average inference time (ms) to process a mini-batch.

We first measure the inference time t_i to process a single mini-batch with different sub-models. Then we measure the *effective accuracy* of SINGA-Easy under the first scenario where only one single model can be loaded to the system.

To measure the actual inference time, we enable GPU warm-up and GPU/CPU synchronization. We also use *torch.cuda.Event* to capture the time before and after model inference. Specifically, we record the inference time with different ingesting rates from 32 to 25,000 instances/second as shown in Figure 7(a). Then the inference time is averaged to obtain the t_i of sub-model m_i. Results in Table 3 show that both the accuracy and inference time decreases with a smaller slice rate, which is consistent with the previous discussion.

To measure the *effective accuracy* of SINGA-Easy, we set the deadline constraint to $D = 8s$ and gradually increase the number of ingested instances N from 32 to 30,000. As shown in Figure 7(b), the model equipped with the scheduler can adapt to the workload by switching between sub-models, which leads to higher *effective accuracy*. Specifically, when $W_{exp} = 100$, the serving model is the full model (i.e., the slice rate $r = 1.0$). When $W_{exp} = 10,000$, the serving model is switched to the smaller model of a slice rate 0.5 to avoid dropping instances. When $W_{exp} = 18,000$, where even the smallest sub-model of a slice rate 0.25 can not process all instances within the time limit. In such scenarios, the serving model is switched to the smallest sub-model to maximize the throughput. As shown in Figure 8(a), the system dynamically adapt the model size to increase the throughput until it reaches the maximum throughput, which is the same as the fastest sub-model.

We also measure the latency of the system, which is shown in Figure 8(b). Specifically, when the ingesting rate is low, since all sub-models now can meet the deadline, the scheduler will adopt

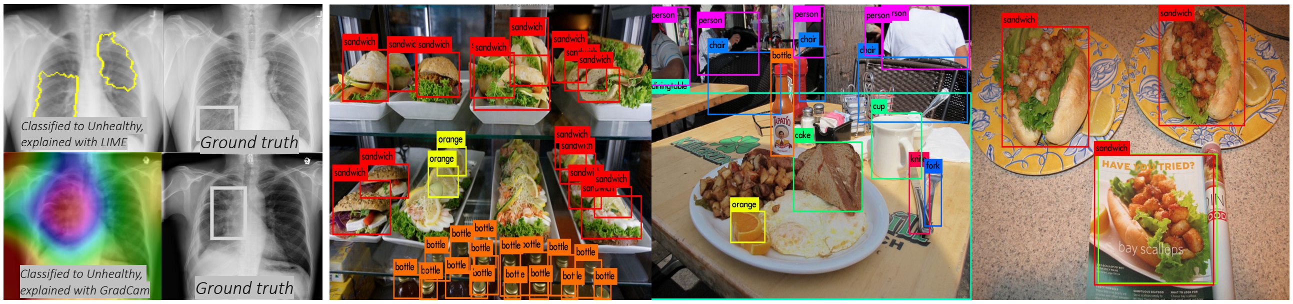


Figure 9: Two multimedia applications(X-ray based diagnosis and Food Detection) developed using SINGA-Easy. The correctness of X-ray explanation is confirmed by the overlap between the explanation map and the ground truth.

the sub-model of slice-rate 1.0 for higher accuracy. When the ingesting rate reach around 10,000, both the sub-models of a slice-rate 1.0 and 0.75 cannot process all instances before the deadline. The combination of sub-models however, can meet the deadline constraint until the ingesting rate reach 18,000, where the serving model will entirely switch to the sub-model of a slice rate 0.25.

To better illustrates the combinations of the scheduled sub-models under different instance ingesting rates, we present detailed assignment of the mini-batches to the sub-models in Figure 8(c). We can observer that when the ingesting rate is low, the model assigns all mini-batches to the sub-model of a slice rate 1.0. Since the sub-model of a slice rate 0.75 and 0.5 have similar accuracy, while the sub-model of a slice rate 0.5 is much faster, the scheduling algorithm tends to use the later sub-model for achieving higher *effective accuracy*.

For the second scenario, where multiple models can be loaded to the system, SINGA-Easy can have multiple elastic models and can generate multiple combinations of sub-models. In contrast, Model-Switching can only have fixed combinations of models.

In conclusion, the experiments on *effective accuracy*, throughput, latency, and sub-model combinations confirm that the model trained with the model slicing technique and our proposed scheduling algorithm support dynamic workloads via finer-grained elastic computation control. It further illustrates the adaptability of SINGA-Easy.

5.3 Multimedia Applications

We further demonstrate the usability of SINGA-Easy on various applications. Due to the space limit, we showcase representative examples in Figure 9. The Singapore Food Detection component has been used to develop FoodLG app[6], which is customized for healthcare applications such as pre-diabetes management and diet recommendation. For the training dataset, we crowdsource to knowledge users using CDAS[26] for labelling. Medical applications like X-ray-based diagnosis is shown in Figure 9, the GradCam map highlights the unhealthy areas with warm colors (red and purple). The LIME map circles the unhealthy areas with yellow color. The explanation maps can assist clinicians in verifying the correctness of the diagnosis, e.g., whether explanation maps match is in line with their diagnosis.

6 RELATED WORK

In this section, we review the related work of ML/DL systems and framework. Their are highly accessible and could be used to extend our SINGA-Easy.

PyTorch [31] can achieve automated ML using the Auto PyTorch library [49], but it does not provide the system infrastructure for ML life cycle management in multimedia applications. SINGA-Easy can be used to facilitate the PyTorch models.

Microsoft NNI[7] is a ML framework supporting model compression. However, it does not provide elastic inference capabilities to the models. While the slice-rate in SINGA-Easy is more understandable.

Hopswork [19] is a data science platform for the design and operation of data analytics applications. The system applies HopsFS, a highly scalable distributed file system, to improve its efficiency. while our system focuses more on the usability to AI applications.

In summary, there are indeed many data analytics systems developed in recent years. Our SINGA-Easy is designed to improve the usability and adaptability in developing multimedia applications.

7 CONCLUSIONS

In this paper, we introduced SINGA-Easy - a learning system focusing on usability and adaptability. SINGA-Easy was built on top of Apache SINGA. It assists users in managing data and models, and developing AI applications. We have used SINGA-Easy to develop multi-media applications such as a chest X-ray image explanation function and food detection system. We showed that SINGA-Easy is highly extendable as it can be used with various third-party machine learning models.

Moving forward, we note that there exist other bottlenecks in data science such as data loading, visualization, cleaning, labeling, and data transformation. Future extensions to SINGA-Easy may include such data science supporting modules.

Acknowledgement: We thank the anonymous reviewers for their constructive comments and NUS colleagues for their comments and contributions. This research is supported by Singapore Ministry of Education Academic Research Fund Tier 3 under MOE's official grant number MOE2017-T3-1-007. Meihui Zhang's work is supported by the National Natural Science Foundation of China (62050099).

[6]http://foodlg.com/

[7]Microsoft NNI: https://github.com/microsoft/nni

REFERENCES

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek Gordon Murray, Benoit Steiner, Paul A. Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2016. TensorFlow: A System for Large-Scale Machine Learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. USENIX Association, Savannah, GA, USA, 265–283.

[2] Dario Amodei, Sundaram Ananthanarayanan, Rishita Anubhai, Jingliang Bai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Jingdong Chen, Mike Chrzanowski, Adam Coates, Greg Diamos, Erich Elsen, Jesse H. Engel, Linxi Fan, Christopher Fougner, Awni Y. Hannun, Billy Jun, Tony Han, Patrick LeGresley, Xiangang Li, Libby Lin, Sharan Narang, Andrew Y. Ng, Sherjil Ozair, Ryan Prenger, Sheng Qian, Jonathan Raiman, Sanjeev Satheesh, David Seetapun, Shubho Sengupta, Chong Wang, Yi Wang, Zhiqian Wang, Bo Xiao, Yan Xie, Dani Yogatama, Jun Zhan, and Zhenyao Zhu. 2016. Deep Speech 2: End-to-End Speech Recognition in English and Mandarin. In *Proceedings of the 33rd International Conference on Machine Learning (ICML)*. ACM, New York, NY, USA, 173–182.

[3] Léon Bottou. 2010. Large-Scale Machine Learning with Stochastic Gradient Descent. In *19th International Conference on Computational Statistics (COMPSTAT)*. Springer, Paris, France, 177–186.

[4] Léon Bottou. 2012. Stochastic Gradient Descent Tricks. In *Neural Networks: Tricks of the Trade - Second Edition*. Springer, Heidelberg, Berlin, 421–436.

[5] Shaofeng Cai, Gang Chen, Beng Chin Ooi, and Jinyang Gao. 2019. Model Slicing for Supporting Complex Analytics with Elastic Inference Cost and Resource Constraints. *Proceedings of the VLDB Endowment* 13, 2 (2019), 86–99.

[6] Shaofeng Cai, Jinyang Gao, Meihui Zhang, Wei Wang, Gang Chen, and Beng Chin Ooi. 2019. Effective and Efficient Dropout for Deep Convolutional Neural Networks. *CoRR* abs/1904.03392 (2019). arXiv:1904.03392 http://arxiv.org/abs/1904.03392

[7] Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A Scalable Tree Boosting System. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, San Francisco, CA, USA, 785–794.

[8] Daniel Crankshaw, Xin Wang, Giulio Zhou, Michael J. Franklin, Joseph E. Gonzalez, and Ion Stoica. 2017. Clipper: A Low-Latency Online Prediction Serving System. In *14th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2017, Boston, MA, USA, March 27-29, 2017*. USENIX Association, 613–627.

[9] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Fei-Fei Li. 2009. ImageNet: A large-scale hierarchical image database. (2009), 248–255. https://doi.org/10.1109/CVPR.2009.5206848

[10] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*. NAACL, Minneapolis, MN, USA, 4171–4186.

[11] Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Tobias Springenberg, Manuel Blum, and Frank Hutter. 2015. Efficient and Robust Automated Machine Learning. In *Advances in Neural Information Processing Systems: Annual Conference on Neural Information Processing Systems*. MIT Press, Montreal, Quebec, Canada, 2962–2970.

[12] Kenneth R Foster, Robert Koprowski, and Joseph D Skufca. 2014. Machine learning, medical diagnosis, and biomedical engineering research-commentary. *Biomedical engineering online* 13 (2014), 94.

[13] Matt Gardner, Joel Grus, Mark Neumann, Oyvind Tafjord, Pradeep Dasigi, Nelson F. Liu, Matthew E. Peters, Michael Schmitz, and Luke Zettlemoyer. 2018. AllenNLP: A Deep Semantic Natural Language Processing Platform. *CoRR* abs/1803.07640 (2018), 6 pages.

[14] Randy Goebel, Ajay Chander, Katharina Holzinger, Freddy Lécué, Zeynep Akata, Simone Stumpf, Peter Kieseberg, and Andreas Holzinger. 2018. Explainable AI: The New 42?. In *Machine Learning and Knowledge Extraction - Second IFIP International Cross-Domain Conference*. Springer, Hamburg, Germany, 295–303.

[15] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross B. Girshick. 2017. Mask R-CNN. In *IEEE International Conference on Computer Vision (ICCV)*. IEEE, Venice, Italy, 2980–2988.

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, Las Vegas, NV, USA, 770–778.

[17] Tin Kam Ho. 1995. Random decision forests. In *Third International Conference on Document Analysis and Recognition (ICDAR)*. IEEE, Montreal, Canada, 278–282.

[18] Andreas Holzinger, Chris Biemann, Constantinos S. Pattichis, and Douglas B. Kell. 2017. What do we need to build explainable AI systems for the medical domain? *CoRR* abs/1712.09923 (2017), 28 pages. http://arxiv.org/abs/1712.09923

[19] Mahmoud Ismail, Ermias Gebremeskel, Theofilos Kakantousis, Gautier Berthou, and Jim Dowling. 2017. Hopsworks: Improving User Experience and Development on Hadoop with Scalable, Strongly Consistent Metadata. In *37th IEEE International Conference on Distributed Computing Systems (ICDCS)*. IEEE, Atlanta, GA, USA, 2525–2528.

[20] Haifeng Jin, Qingquan Song, and Xia Hu. 2019. Auto-Keras: An Efficient Neural Architecture Search System. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD)*. ACM, Anchorage, AK, USA, 1946–1956.

[21] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *3rd International Conference on Learning Representations (ICLR)*. ICLR Press, San Diego, CA, USA, 15 pages.

[22] Lars Kotthoff, Chris Thornton, Holger H. Hoos, Frank Hutter, and Kevin Leyton-Brown. 2017. Auto-WEKA 2.0: Automatic model selection and hyperparameter optimization in WEKA. *Journal of Machine Learning Research* 18 (2017), 25:1–25:5.

[23] Alex Krizhevsky and Geoffrey Hinton. 2009. Learning multiple layers of features from tiny images. (2009).

[24] Erin LeDell and Sebastien Poirier. 2020. H2O AutoML: Scalable Automatic Machine Learning. In *7th ICML Workshop on Automated Machine Learning (AutoML)*. ACM, Virtual Conference, 16 pages.

[25] Sijia Liu, Parikshit Ram, Deepak Vijaykeerthy, Djallel Bouneffouf, Gregory Bramble, Horst Samulowitz, Dakuo Wang, Andrew Conn, and Alexander G. Gray. 2020. An ADMM Based Framework for AutoML Pipeline Configuration. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence (AAAI)*. AAAI, New York, NY, USA, 4892–4899.

[26] Xuan Liu, Meiyu Lu, Beng Chin Ooi, Yanyan Shen, Sai Wu, and Meihui Zhang. 2012. CDAS: A Crowdsourcing Data Analytics System. *Proc. VLDB Endow.* 5, 10 (2012), 1040–1051. https://doi.org/10.14778/2336664.2336676

[27] Zhaojing Luo, Shaofeng Cai, Jinyang Gao, Meihui Zhang, Kee Yuan Ngiam, Gang Chen, and Wang-Chien Lee. 2018. Adaptive lightweight regularization tool for complex analytics. In *IEEE 34th International Conference on Data Engineering (ICDE)*. 485–496.

[28] Jorge G. Madrid, Hugo Jair Escalante, Eduardo F. Morales, Wei-Wei Tu, Yang Yu, Lisheng Sun-Hosoya, Isabelle Guyon, and Michèle Sebag. 2019. Towards AutoML in the presence of Drift: first results. *CoRR* abs/1907.10772 (2019), 14 pages. http://arxiv.org/abs/1907.10772

[29] Beng Chin Ooi, Kian-Lee Tan, Sheng Wang, Wei Wang, Qingchao Cai, Gang Chen, Jinyang Gao, Zhaojing Luo, Anthony K. H. Tung afnd Yuan Wafng, Zhongle Xie, Meihui Zhang, and Kaiping Zheng. 2015. SINGA: A Distributed Deep Learning Platform. In *Proceedings of the 23rd Annual ACM Conference on Multimedia Conference*. ACM, Brisbane, Australia, 685–688.

[30] Akin Özçift and Arif Gülten. 2011. Classifier ensemble construction with rotation forest to improve medical diagnosis performance of machine learning algorithms. *Computer Methods and Programs in Biomedicine* 104, 3 (2011), 443–451.

[31] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems*. MIT Press, Vancouver, BC, Canada, 8024–8035.

[32] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake VanderPlas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Edouard Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *J. Mach. Learn. Res.* 12 (2011), 2825–2830.

[33] Martin Pelikan, David E Goldberg, Erick Cantú-Paz, et al. 1999. BOA: The Bayesian optimization algorithm. In *Proceedings of the genetic and evolutionary computation conference (GECCO)*. ACM, Orlando, Florida, USA, 525–532.

[34] Vitali Petsiuk, Abir Das, and Kate Saenko. 2018. RISE: Randomized Input Sampling for Explanation of Black-box Models. In *British Machine Vision Conference 2018 (BMVC)*. British Machine Vision Association, Newcastle, UK, 151.

[35] Joseph Redmon and Ali Farhadi. 2018. YOLOv3: An Incremental Improvement. *CoRR* abs/1804.02767 (2018). arXiv:1804.02767 http://arxiv.org/abs/1804.02767

[36] Marco Túlio Ribeiro, Sameer Singh, and Carlos Guestrin. 2016. "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*. ACM, San Francisco, CA, USA, 1135–1144.

[37] Yuji Roh, Geon Heo, and Steven Euijong Whang. 2021. A Survey on Data Collection for Machine Learning: A Big Data - AI Integration Perspective. *IEEE Transactions on Knowledge and Data Engineering* 33, 4 (2021), 1328–1347.

[38] Ramprasaath R. Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. 2017. Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization. In *IEEE International Conference on Computer Vision (ICCV)*. IEEE, Venice, Italy, 618–626.

[39] Karen Simonyan and Andrew Zisserman. 2015. Very Deep Convolutional Networks for Large-Scale Image Recognition. In *3rd International Conference on Learning Representations (ICLR)*. ICLR Press, San Diego, CA, USA, 15 pages.

[40] Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. 2012. Practical Bayesian Optimization of Machine Learning Algorithms. In *Advances in Neural Information Processing Systems: 26th Annual Conference on Neural Information Processing Systems*. MIT Press, Lake Tahoe, Nevada, United States, 2960–2968.

[41] Danding Wang, Qian Yang, Ashraf Abdul, and Brian Y Lim. 2019. Designing theory-driven user-centric explainable AI. In *Proceedings of the CHI conference on human factors in computing systems*. ACM, Glasgow, Scotland, UK, 1–15.

[42] Fei Wang, Rainu Kaushal, and Dhruv Khullar. 2020. Should health care demand interpretable artificial intelligence or accept "black box" medicine? *Annals of Internal Medicine* 172, 1 (2020), 59–60.

[43] Wei Wang, Jinyang Gao, Meihui Zhang, Sheng Wang, Gang Chen, Teck Khim Ng, Beng Chin Ooi, Jie Shao, and Moaz Reyad. 2018. Rafiki: Machine Learning as an Analytics Service System. *Proceedings of the VLDB Endowment* 12, 2 (2018), 128–140.

[44] Wei Wang, Xiaoyan Yang, Beng Chin Ooi, Dongxiang Zhang, and Yueting Zhuang. 2016. Effective deep learning-based multi-modal retrieval. *The VLDB Journal* 25, 1 (2016), 79–101.

[45] Wei Wang, Meihui Zhang, Gang Chen, HV Jagadish, Beng Chin Ooi, and Kian-Lee Tan. 2016. Database meets deep learning: Challenges and opportunities. *ACM SIGMOD Record* 45, 2 (2016), 17–22.

[46] Shinji Watanabe, Takaaki Hori, Shigeki Karita, Tomoki Hayashi, Jiro Nishitoba, Yuya Unno, Nelson Enrique Yalta Soplin, Jahn Heymann, Matthew Wiesner, Nanxin Chen, Adithya Renduchintala, and Tsubasa Ochiai. 2018. ESPnet: End-to-End Speech Processing Toolkit. In *Proceedings of the 19th Annual Conference of the International Speech Communication Association (Interspeech)*. Elsevier, Hyderabad, India, 2207–2211.

[47] Yang You, Zhao Zhang, Cho-Jui Hsieh, James Demmel, and Kurt Keutzer. 2018. ImageNet Training in Minutes. In *Proceedings of the 47th International Conference on Parallel Processing (ICPP)*. ACM, Eugene, OR, USA, 1:1–1:10.

[48] Jeff Zhang, Sameh Elnikety, Shuayb Zarar, Atul Gupta, and Siddharth Garg. 2020. Model-Switching: Dealing with Fluctuating Workloads in Machine-Learning-as-a-Service Systems. In *12th USENIX Workshop on Hot Topics in Cloud Computing, HotCloud 2020, July 13-14, 2020*, Amar Phanishayee and Ryan Stutsman (Eds.). USENIX Association.

[49] Lucas Zimmer, Marius Lindauer, and Frank Hutter. 2020. Auto-PyTorch Tabular: Multi-Fidelity MetaLearning for Efficient and Robust AutoDL. *CoRR* abs/2006.13799 (2020), 1–15.