Privacy-preserving Online AutoML for Domain-Specific Face Detection

Chenqian Yan[1†*] Yuge Zhang[1†] Quanlu Zhang[1] Yaming Yang[1]
Xinyang Jiang[1] Yuqing Yang[1] Baoyuan Wang[2]
Microsoft Research[1] Xiaobing.ai[2]
im.cqyan@gmail.com,{yugzhan,quzha,yayaming,xinyangjiang,yuqyang}@microsoft.com,wangbaoyuan@xiaobing.ai

Abstract

Despite the impressive progress of general face detection, the tuning of hyper-parameters and architectures is still critical for the performance of a domain-specific face detector. Though existing AutoML works can speedup such process, they either require tuning from scratch for a new scenario or do not consider data privacy. To scale up, we derive a new AutoML setting from a platform perspective. In such setting, new datasets sequentially arrive at the platform, where an architecture and hyper-parameter configuration is recommended to train the optimal face detector for each dataset. This, however, brings two major challenges: (1) how to predict the best configuration for any given dataset without touching their raw images due to the privacy concern? and (2) how to continuously improve the AutoML algorithm from previous tasks and offer a better warm-up for future ones? We introduce "HyperFD", a new privacy-preserving online AutoML framework for face detection. At its core part, a novel meta-feature representation of a dataset as well as its learning paradigm is proposed. Thanks to HyperFD, each local task (client) is able to effectively leverage the learning "experience" of previous tasks without uploading raw images to the platform; meanwhile, the meta-feature extractor is continuously learned to better trade off the bias and variance. Extensive experiments demonstrate the effectiveness and efficiency of our design.

1. Introduction

Face detection [6, 28, 51, 68, 69] is one of the most fundamental problems in computer vision. Although, rapid progress has been made lately for the general cases, bespoken face detection models are still in high-demand for domain-specific scenarios. This is because, the challenges for detecting faces from an outdoor surveillance camera might be different from a panoramic indoor fish-eye camera [13]; likewise, the challenges for detecting occluded faces (*e.g.*, masks [21]) are also quite different from selfie faces captured by cellphone cameras [30]. Therefore, extensive manual parameter tuning and large computing resource is required in order to obtain the best specialized model for each domain. To scale up the scenarios, we see a clear industry demand of building shared AI model training platform so as to leverage pretrained representation from other relevant tasks. For example, Microsoft Custom Vision [34] can train specialized object detection models given a set of user-uploaded images for engineers without deep learning background. This, however, comes at the cost of sacrificing the face data privacy, for face detection models. Similarly, other AutoML tools (*e.g.*, NNI [35]) either do not consider the data privacy or still require tuning from scratch for a new scenario, which is not secure and scalable.

The training of domain-specific face detection for real-world scenarios requires a new problem setup from a platform perspective, where the platform receives new

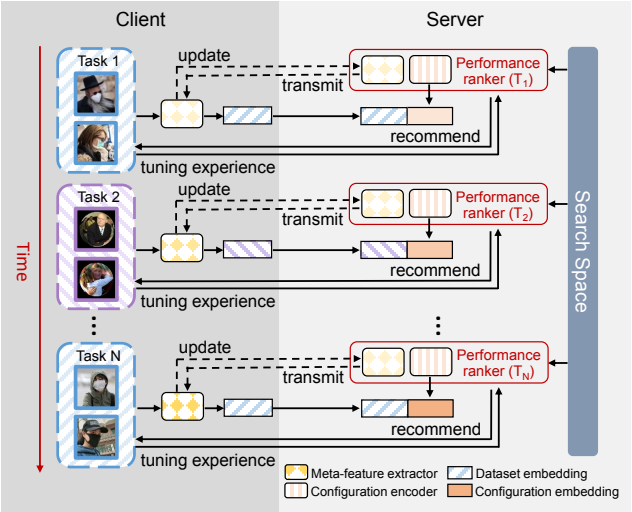

Figure 1. Overview of HyperFD framework, which aims to build a shared AutoML platform that enables exchanging tuning experience among customers, without access to customers' raw datasets. The performance ranker consists of meta-feature extractor and configuration encoder. Both are continuously updated to incorporate tuning experience on the latest task.

*Work done as an intern at MSRA. † Equal contribution.

datasets sequentially, and recommends architecture and hyper-parameter configuration to train the optimal face detector for each dataset (corresponds one particular domain). This problem setting brings two major challenges. The first is to effectively predict the best configuration for any incoming dataset under the constraints that the privacy of their raw images are protected. The second is to leverage the "experience" from previous tasks[1] to continuously improve the AutoML algorithm, such that the platform can better serve future ones.

To tackle the challenge of privacy protection, we derive a new online AutoML paradigm for face detection, which is called "HyperFD". Specifically, instead of uploading the raw images to the server, each local client only sends the dataset level representations (called "meta-features" in the following) to the platform and asks for the best configuration including a network architecture and hyper-parameters. The meta-feature is designed to encode the overall statistics and general attributes of a given dataset. The platform maintains a learnable performance ranker that selects top-k optimal training configurations from the hyper-parameter/architecture search space based on the dataset meta-feature. Finally, after obtaining the configurations from the platform, the face detector is then trained locally on each client. In this way, the platform only sees dataset meta-features and the testing performance, which effectively protect the training data privacy. Figure 1 gives an overview of our HyperFD framework. Note that, although it is federated, HyperFD conducts the actual training task locally and there is no global aggregation is needed, which is different from traditional federated learning [32].

To tackle the second challenge and make HyperFD more generalizable for unseen scenarios, we ask the meta-feature to be updated continuously with the new dataset, yet properly borrow the "experience" from previously trained tasks. Due to the fact that there is no way to access the raw data, we integrate a novel meta-feature transformation module that builds a mapping between the current meta-feature space and previous feature space. Intuitively, this mapping will help make similar distributed historical tasks play more influence in ranking the final configuration for the new task. To summarize, we make the following contributions:

- We introduce privacy-preserving online AutoML for face detection, which is a new problem setting from platform perspective.
- We propose a novel meta-feature extractor to build better dataset level representations, which is trained continuously without touching the raw face images.
- Extensive experiments show the superior performance of our approach. We will also release the benchmark and source code to facilitate future research.

[1] In this paper, we use "task" and "dataset" interchangeably.

2. Related works

Transferable AutoML. Early works of transferable AutoML addressed the problem from a multi-task collaborative tuning perspective [2, 15, 47], in the hope that running multiple AutoML tasks simultaneously will help each other achieve better results. The underlying techniques include surrogate-based ranking [2], multi-task Gaussian processes [47] and probabilistic matrix factorization [15]. A later stream of researches focus on the "warm-start" setting, which is to recommend a good starting point for a new AutoML task based on previous experiments [12, 25, 36, 60]. However, from a platform perspective, a more realistic setting would be, training tasks arrive sequentially and one needs to search the best configuration for a coming dataset. Xue *et al*. [59] firstly refers to this setting as "AutoML under *online setting*". Although a few attempts have been made to formulate and solve such problem [55, 59], they still neglected the essential constraints of privacy and assumed all datasets to be directly accessible to the algorithm, which is infeasible especially for sensitive datasets like face.

Dataset meta-feature. Dataset meta-feature (or representation) is shown to be crucial to the performance of transferable AutoML [23]. The most straight-forward and earliest dataset representation is based on descriptive statistics of a dataset [2, 11, 24, 33, 62], *e.g*., number of images. More advanced approaches include the usage of pretrained models' performances on unseen datasets [12, 54, 59] along with their landscape [1]. However, these methods are heuristically designed and not directly aware of the end-to-end goal of AutoML. An alternative way is to optimize deep features of neural network [23, 25, 36, 53] in an end-to-end manner. Although effective, this has not become the mainstream and recent works targeting at online setting [55, 59] are still using heuristically designed representations. Moreover, most works are limited on image classification and tabular dataset for their evaluation. To the best of our knowledge, we are the first to extract meta-features for face detection datasets.

Continual learning. Continual learning is a scenario where a single neural network needs to sequentially learn a series of tasks. The crucial challenge is catastrophic forgetting that parameters or semantics learned for the past tasks drift to the direction of new tasks. To alleviate such issue, regularization on gradients [26, 67], designs of dynamic architectures [56, 64, 66], replay of previous training data [27, 41, 45] are usually needed. Despite the various techniques, joint training on all history data is still considered the upper bound for continual learning [48]. Recent researches proposed federated continual learning [22, 63], putting privacy into consideration, but they are essentially different from us. Their clients aggregate parameters, while we propose to share meta-level knowledge of AutoML experience. Moreover, their solution is designed for Task-IL scenario [48] and not applicable to ours.

3. HyperFD

3.1. Framework overview

Figure 1 illustrates an overview of our HyperFD framework, where the domain-specific training tasks for face detection sequentially arrive. For a new task, meta-feature extractor maintained on the server is transmitted to the client. The client extracts features from the face dataset with the extractor, and sends the features back to the server without disclosing the raw images. The server maintains a search space which has various model architectures and hyper-parameters for face detection. A performance ranker predicts the rank of the configurations on that client's task based on its meta-feature and suggests several configurations to the client. The client verifies the performance of those configurations on its dataset, and informs the performance to the server. Then the server updates the performance ranker with the newly collected data. This is a continual learning process with tuning experience on new face detection tasks continuously arriving. The problem is that raw datasets are not available on the server and every client becomes unreachable after its task is done. Thus, we design a meta-feature transformation module. It continuously projects the features extracted with old-version extractors to the feature space of the latest extractor, so that the updates could leverage all the historical data to prevent forgetting. The loss function combines ranking loss, regularization, and synaptic intelligence to guarantee steady improvement of the whole framework.

3.2. Performance ranker

Performance ranker is the basic building block of HyperFD framework. It ranks configurations from a search space for each dataset. The search space $\mathcal{C}$ contains $|\mathcal{C}| = M$ different configurations, the k-th of which is denoted as c_k. We use $\mathcal{D} = \{d_1, d_2, \ldots, d_N\}$ to denote the datasets of face detection tasks, where d_t corresponds to the t-th task. Our goal is to learn a performance ranker that, for any pair of c_k and d_t, it predicts a score (*e.g.*, AP@50).

We formulate the ranker as a differentiable function $F(c_k, d_t; \theta_F)$ parameterized by θ_F. F consists of two key components: a configuration encoder $H(c_k; \theta_H)$ that maps any configuration in $\mathcal{C}$ into a fixed-length vector, and a dataset meta-feature extractor $G(d_t; \theta_G)$ that extracts semantic information from a dataset to generate a fixed-length vector. G and H are learned such that the configuration embedding and dataset embedding lie in the same embedding space, and we measure their similarity with an inner product. A higher similarity means a better match of an configuration and a dataset, leading to a potentially better performance.

$$F(c_k, d_t; \theta_F) = G(d_t; \theta_G)^\intercal H(c_k; \theta_H) \tag{1}$$

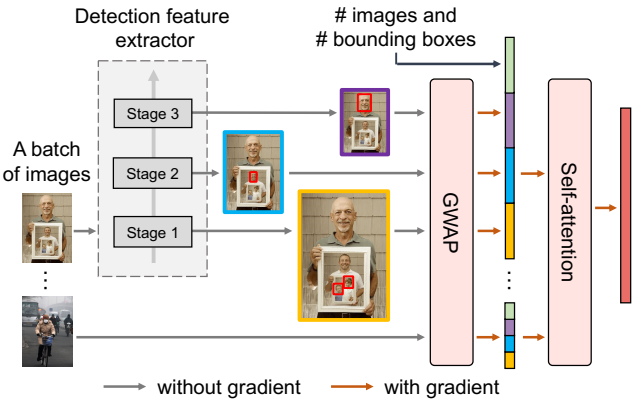

Figure 2. Architecture of meta-feature extractor.

In our framework, the performance ranker is optimized in a supervised learning manner on *tuning experience*. Here, tuning experience is a meta-dataset consisting of a number of triplets $\{(c_{u_i}, d_{v_i}, a_i)\}_{i=1}^{S}$ $(1 \le u_i \le M, 1 \le v_i \le N)$, where u_i is the index of the configuration evaluated on the v_i-th dataset in the i-th triplet, a_i is its performance, *i.e.*, evaluation score of a detector trained with c_{u_i} on d_{v_i}.

3.2.1 Meta-feature extractor

The key challenge to design a performance ranker is how to effectively extract meta-features from a dataset. Inspired by previous works on image classification [25, 36], we follow the intuition that semantic information from the raw images (*e.g.*, statistical information of images and annotations, vision features extracted with a pretrained model) could decently convey the characteristics of a dataset. However, face detection is more complex, which spans multi-scale anchors combined with per-anchor classification and regression. Thus, we design a novel hierarchical feature extraction approach starting from anchor-level, followed by a series of aggregations to generate a high-level dataset feature. The overall architecture is shown in Figure 2.

Anchor-level. To handle raw images, we first feed them into a pretrained face detector, RetinaFace [6]. We use the feature pyramid generated by context modules (*i.e.* SSH [38]), which consists of three feature maps downsampled by 8, 16 and 32 respectively. These feature maps are responsible for detecting small, medium, and large faces in the original detector design. We call those feature maps anchor-level features because every pixel on the feature map corresponds to one or several anchors in detection tasks. Apart from features extracted by detector, we also attach the matching ground-truth bounding boxes to each anchor to enrich the information. The feature map for the k-th stage is denoted with $\text{det}^k(I)$, whose height is H_k and width is W_k, where I is a preprocessed image,

Image-level. To aggregate anchor-level features into

image-level, each of the three feature maps is fed into a Global Weighted Average Pooling (GWAP) [40], which in our case is to balance the weights for unbalanced positive and negative samples in face detection. We reweight the anchors such that different levels of IoU matching ratios contribute equally to the results. Concretely, for $k = 1, 2, 3$, we group the anchors on $\text{det}^k(I)$ according to levels of IoU matching ratios into positive anchors, negative anchors, and ignored anchors [4], and average the anchors within groups before a total average:

$$\overline{\text{det}}^k(I) = \frac{1}{3}\left(\overline{\text{det}}^k_{\text{pos}}(I) + \overline{\text{det}}^k_{\text{neg}}(I) + \overline{\text{det}}^k_{\text{ign}}(I)\right) \quad (2)$$

where $\overline{\text{det}}^k_{\text{pos}}$, $\overline{\text{det}}^k_{\text{neg}}$, $\overline{\text{det}}^k_{\text{ign}}$ are the average of the stage-k feature map over positive, negative and ignored anchors respectively. The feature for one image is then a concatenation of $\overline{\text{det}}^k(I)$ for $k = 1, 2, 3$. Another vector of high-level descriptive statistics of the dataset [62] is concatenated, which, in our case, are number of images and bounding boxes of the whole dataset. This is needed because such information is missing in the context of a batch of images. The information is injected at the image level so that the follow-up modules can fuse it with other visual semantics.

Dataset-level. This level aggregates the features of all the images of a dataset to capture the distribution of the images' features. We propose to use self-attention (*i.e.*, a transformer encoder layer [50]) to extract the distribution semantic. The rationale behind this is that the optimal configuration suited for one task is mostly correlated to the distribution of the dataset. Specifically, positional embedding is not used because the sequence of images is permutation invariant. After the features fused among images, an average pooling is used to obtain a dataset-level feature, *i.e.*, meta-feature of the dataset.

3.2.2 Configuration encoder

Different types of search spaces prefer different configuration encoders. For a hyper-parameter search space which consists of one or more categorical variables, Multi-layer Perceptron is a simple yet effective choice. For neural architecture search space, a more sophisticated encoder is desired, so as to handle the complexity in neural networks. There is recently a growing popularity to use Graph Neural Networks as the performance predictor in NAS [7, 46, 52], because neural networks are essentially graphs. To be specific, we use GIN [57] as the configuration encoder for neural architectures due to its superiority over other GNNs [46].

3.3. Privacy-preserving continual learning

In HyperFD, performance ranker runs in a scenario that raw datasets on the client side are not accessible by the server and the client is only reachable during its own task. However, continuously improving performance ranker requires joint training on experience of both incoming new tasks and old ones. This brings two new challenges to continual learning. Firstly, the server only stores meta-features rather than raw datasets. Secondly, the meta-features are generated with different versions of meta-feature extractor due to continual learning. To address those challenges, we design a novel meta-feature transformation module combined with three loss functions, which guarantees stable and effective update of the performance ranker.

3.3.1 Meta-feature transformation module

Our performance ranker can be naturally decomposed into G and H. G is executed and updated on the client side, thus, preserves privacy of the client. The updated G is sent back to the server. As G keeps evolving along with incoming tasks, we introduce t to denote the t-th task (assuming tasks arrive sequentially). The t-th task has dataset d_t, meta-feature $G^{(t)}(d_t; \theta_G^{(t)})$, where $\theta_G^{(t)}$ is the weight of the meta-feature extractor in the state of finishing t-th task. We use $G^{(t)}(d_t)$ for short in the rest of this paper. The tuning experience triplets on the server is $\{c_{u_i}, \phi_{v_i}, a_i\}_{i=1}^S$, where ϕ_{v_i} the meta-feature of d_{v_i}, defined as follows:

$$\phi_t = \begin{cases} G(d_\tau; \theta_G), & \text{if } t = \tau \\ G^{(t)}(d_t; \theta_G^{(t)}), & \text{otherwise} \end{cases} \quad (3)$$

where d_τ is the dataset of the current (*i.e.* τ-th) task.

From Equation 3 we can see that, the meta-features of current dataset and the meta-features in the past are extracted with different meta-feature extractors, and lie in different feature spaces. The oracle solution would be that we ask the users to extract the meta-feature again with the current extractor, but it is infeasible in our scenario. To align meta-features to the latest feature space, we project the meta-features extracted with old meta-feature extractors to the latest feature space using linear mapping. We call this *meta-feature transformation*.

Assume our system is currently ready to serve dataset d_τ. The latest meta-feature extractor is denoted as G. We aim to predict $G(d_t)$, with $G^{(t)}(d_t)$ as input, multiplied by a transformation matrix $\mathbf{Z}^{(t)}$ (one matrix for each dataset). It becomes a supervised learning problem to minimize the distance between $\mathbf{Z}^{(t)}G^{(t)}(d_t)$ and $G(d_t)$. To learn $\mathbf{Z}^{(t)}$, we need a plenty number of pairs $(G^{(t)}(d), G(d))$, where d is any dataset. It is impossible to collect such data from users, because we cannot expect "plenty" of users to be online when serving a new user. Thus, we take d from $\mathcal{D}_{\text{offline}}$, a series of datasets prepared in HyperFD framework offline, whose raw data are always accessible. Then $\mathbf{Z}^{(t)}$ can be trained via:

$$\mathcal{L}_{\text{trans}}(\mathbf{Z}^{(t)}) = \sum_{d \in \mathcal{D}_{\text{offline}}} \|\mathbf{Z}^{(t)}G^{(t)}(d) - G(d)\|^2 \quad (4)$$

With the transformation modules $\mathbf{Z}^{(1)}, \mathbf{Z}^{(2)}, \ldots, \mathbf{Z}^{(\tau-1)}$ sufficiently trained, we have $\mathbf{Z}^{(t)}G^{(t)}(d) \approx G(d)$ hold for any dataset d. We can then extend the definition of performance ranker F to $\mathcal{F}$, so that it can work on any pair of configurations and meta-features.

$$\mathcal{F}(c_k, \phi_t; \theta_F) = \begin{cases} \phi_t^\intercal H(c_k), & \text{if } t = \tau \\ \left(\mathbf{Z}^{(t)}\phi_t\right)^\intercal H(c_k), & \text{otherwise} \end{cases} \tag{5}$$

In this equation, when updating θ_F with back propagation, θ_G can be updated only for the branch of current dataset d_τ. On the other hand, for $t < \tau$ there is no gradients, because ϕ_t is pre-computed and $\mathbf{Z}^{(t)}$ is already optimized. Hence, the joint training on previous experience is essentially adapting the configuration encoder to the feature space of the latest meta-feature extractor.

3.3.2 Loss functions

Next, we introduce the loss function to optimize the performance ranker.

Ranking loss. Our primary loss function is a ranking loss proposed by [70] that penalizes imperfect ranking of configurations. The loss is calculated on $\{(c_{u_i}, \phi_{v_i}, a_i)\}_{i=1}^K$, and is in the form of,

$$\mathcal{L}_{\text{rank}}(F) = \mathbb{E}_{\substack{v_i=v_j \\ a_i>a_j}}[-\Delta_{\text{NDCG}} \cdot \log \sigma(\mathcal{F}(c_{u_i}, \phi_{v_i}) - \mathcal{F}(c_{u_j}, \phi_{v_j}))] \tag{6}$$

where σ refers to the sigmoid function and Δ_{NDCG} is the changes of a ranking metric, *i.e.*, Normalized Discounted Cumulative Gain (NDCG) [20] in particular, after switching the ranking position of i and j, so that ranking failures on top items are emphasized.

Triplet regularization. As the tuning experience triplets to train the ranker may not be "plenty" enough, it is critical to have a proper regularization. Similar to previous works [23, 36], we use a triplet loss with margin [44] to minimize the distance to a batch from the same dataset minus the distance to another dataset. Formally, it is defined as,

$$\mathcal{L}_{\text{sim}}(G) = \max\left(\|\phi_{v_i} - \widetilde{\phi_{v_i}}\|^2 - \|\phi_{v_i} - \phi_{v_j}\|^2 + \alpha, 0\right) \tag{7}$$

where $v_i \neq v_j$. ϕ_{v_i} and $\widetilde{\phi_{v_i}}$ are the meta-feature of the same dataset but extracted with different representative image samples, ϕ_{v_j} and ϕ_{v_i} are different datasets and α is a hyper-parameter controlling the margin. Note that this loss is only applicable when $v_i = \tau$ or $v_j = \tau$. Otherwise, G will not receive any gradients.

Synaptic Intelligence. Though we used joint training to incorporate the experience from both past and present, as discussed previously, only the configuration encoder is optimized. Although the whole performance ranker benefits from those updates, the meta-feature extractor could still experience forgetting issue, which potentially degrades the performance. Because the meta-feature extractor is federatedly trained on the client side, we do not have access to their raw data. Thus, we adopt another continual learning technique, Synaptic Intelligence (SI) [67], which is a regularization loss that nicely fits our framework. To this end, for any dataset d_t, we first calculate $\omega^{(t)}$, which is a per-parameter contribution to the change of loss:

$$\omega^{(t)} = -\sum_{i=1}^{N_{\text{iters}}} \left(\theta^{(t,i)} - \theta^{(t,i-1)}\right) \odot \frac{\delta \mathcal{L}_{\text{unreg}}^{(t,i)}}{\delta \theta^{(t,i)}} \tag{8}$$

where N_{iters} denotes the number of iterations. $\theta^{(t,i)}$ is the parameters after the i-th iteration of training on dataset d_t. $\mathcal{L}_{\text{unreg}}^{(t,i)}$ represents the loss without this regularization term at the i-th iteration. $\odot$ means element-wise product. We sum ω over the datasets to get $\Omega^{(\tau-1)}$, which the importance of every parameter in the first $\tau-1$ datasets (τ is current time):

$$\Omega_i^{(\tau-1)} = \sum_{t=1}^{\tau-1} \frac{\omega_i^{(t)}}{\left(\theta_i^{(t)} - \theta_i^{(t-1)}\right)^2 + \xi} \tag{9}$$

where θ_i is the i-th parameter in θ. ξ is a small dampening term to prevent dividing by zero, which we set to 0.1. Then, the regularization loss is given by,

$$\mathcal{L}_{\text{reg}}(F) = \sum_{i=1}^{|\theta|} \Omega_i^{(\tau-1)} (\theta_i - \theta_i^{(\tau-1)})^2 \tag{10}$$

Overall, the loss for the performance predictor is,

$$\mathcal{L}_{\text{total}}(F) = \mathcal{L}_{\text{rank}}(F) + \lambda_{\text{sim}}\mathcal{L}_{\text{sim}}(G) + \lambda_{\text{reg}}\mathcal{L}_{\text{reg}}(F) \tag{11}$$

λ_{sim} and λ_{reg} are hyper-parameters controlling the weight of regularizations.

4. Experiments

4.1. Experiment setup

Face detector training. We use RetinaFace [6] with MobileNet-V2 [43] backbone, pretrained on WIDER-Face [61]. To train on a new dataset, we inherit weights pretrained on WIDER-Face, and fine-tune it on the target dataset. If the model used a different architecture from the pre-trained model, we perform a network adaptation with parameter remapping [10]. We adopt "Reduce LR on Plateau" learning rate scheduler to ensure convergence. For evaluation, we follow [6,29,58] to rescale the shorter side of images to 720 pixels. We use Average-Precision at IoU 0.5 (AP@50) as our evaluation metric. A more detailed training setup is provided in Appendix B.1.

Datasets. We evaluate HyperFD on 12 publicly available face datasets: AFLW [31], Anime [39], Face-Mask [18], FDDB [19], FDDB-360 [13], MAFA [16],

Pascal VOC [8], UFDD [37], UMDAA-02 [30], WIDER-Face [61], WIDER-360 [14], WIKI [42]. Details on data cleaning and train/val/test split can be found in Appendix B.2. We split the datasets into server side and client side, where WIDER-Face is considered always available at the server side. The rest of the datasets are treated as customer data which is not directly visible to the central server. We shuffle the order of the 11 customer datasets before every experiment, so as to simulate the scenario of customers' data coming by in an arbitrary order.

Search space. We define two search spaces for our evaluation. (i) **HPO space** (*i.e*. hyper-parameter search space) tunes 6 different dimensions and contains 216 combinations of hyper-parameters, spanning from generic hyperparameters (*e.g*., learning rate) to domain-specific ones (*e.g*., IoU threshold). (ii) **NAS space** tunes the backbone architecture, as backbones are found to be essential to the detection performance [5]. Our backbone search space is a MobileNetV2-like space proposed by [3]. We limit the FLOPs to be less than 730M under 360P resolution to avoid selecting extra large models. The size of whole search space is $2.44 \cdot 10^{15}$. Details available in Appendix B.3.

Dataset augmentation. With only WIDER-Face available at the beginning, it is difficult to obtain a meaningful meta-feature extractor. Also, the training of meta-feature transformation module relies on a diversity of datasets so that it does not overfit to a particular representation. Following [55,59], we extract subsets from WIDER-Face to create a variety of datasets on the server side. We intentionally manipulate the distribution of each subset to increase diversity in domains. This is done by clustering features generated by different deep learning models. We end up creating 1418 datasets to form $\mathcal{D}_{\text{offline}}$. Refer to Appendix B.4 for details.

Performance ranker. For each of HPO and NAS space, we sample 3,000 pairs of configurations and datasets (from $\mathcal{D}_{\text{offline}}$), and get their corresponding performance. This forms a meta-dataset which we use to "*warm-up*" the ranker before online datasets are received. During online stage, for each dataset, we randomly sample 200 configurations (that almost exhausts the space for HPO), and get B configurations according to the prediction scores. An exploration-exploitation strategy [7] is adopted to prevent the selected configurations becoming too homogeneous. Following [36], we pre-built a performance *benchmark* (*i.e*., lookup table), so that we do not have to go through the computationally-expensive step of detector training in every experiment. Budget, *i.e*., the number of trials for each dataset, is set to 4 if not otherwise specified. Other hyper-parameters can be found in Appendix B.5.

4.2. Performance of HyperFD

Baselines. We compare our method with the following baselines: (i) **Random search**: Randomly selecting certain number of configurations, without any knowledge over dataset or search space. (ii) **Best on WIDER**: Finding the top configurations on WIDER-Face (by grid search on HPO space and performance predictor on NAS space) and applying them on new datasets. (iii) **Tr-AutoML** [59]: A Markov-analysis based method. It can be applied to our scenario because it is a transferable AutoML algorithm that is designed for an online setting similar to ours. (iv) **HyperSTAR** [36]: The framework is optimized in an end-to-end manner. It adopts a frozen ResNet50 [17] as meta-feature extractor, and thus can be adapted to our scenario. We fine-tune its predictor on all historic experience before serving new datasets. (v) **SCoT** [2]: A bayesian optimization framework whose surrogate model learns to predict performance conditioned jointly on descriptive statistics of datasets and configurations.

Method	HPO space		NAS space	
	ΔAP (↑)	Rank (↓)	ΔAP (↑)	Rank (↓)
Random search	0.00	20.09	0.00	20.10
Best on WIDER	1.33	25.12	2.04	10.97
Tr-AutoML [59]	-0.18	22.18	-0.52	25.29
HyperSTAR [36]	-0.47	22.00	-0.14	21.09
SCoT [2]	-0.10	20.83	-0.28	21.30
HyperFD (ResNet)	-0.23	17.41	1.54	11.97
HyperFD (Statistics)	0.08	17.59	1.58	12.06
HyperFD (MSE)	-0.05	20.93	0.21	18.34
HyperFD	**1.67**	**13.16**	**2.39**	**7.78**

Table 1. Comparison of end-to-end performance with baselines and several variants. **HyperFD (ResNet)**: using ResNet50 as meta-feature extractor. **HyperFD (Statistics)**: using descriptive statistics as meta-features. **HyperFD (MSE)**: using MSE as the primary loss.

Evaluation metrics. The metrics used in our evaluation are: (i) **ΔAP**: The performance gain of the proposed search algorithm compared to random search under the same budget. The gains are *summed* across datasets. The higher the better (↑). (ii) **Rank**: The rank of the best found configuration in the search space. The rank is normalized to 0∼100% for easy comparison across different search spaces, then averaged over datasets. The lower the better (↓).

Results. We show the results in Table 1. We evaluate each setting with 20 different seeds and report the average. The standard deviations along with more detailed results on each dataset can be found in Appendix C.

We firstly point out that we cannot rely on the tuning experience of one single dataset. The top configurations on WIDER-Face rank 25.12% and 10.97% in average for HPO and NAS search space respectively, indicating that no golden configuration works on all datasets. Although on some datasets, the best configuration on WIDER-Face can be impressive (*e.g*., WIDER-360, HPO, +2.5% ΔAP), for some datasets they can be almost at the bottom (*e.g*., ANIME, HPO, worse than 85.6% of the search space).

Surprisingly, in our scenario, the sophisticated approaches (*e.g*., HyperSTAR) perform even worse than a

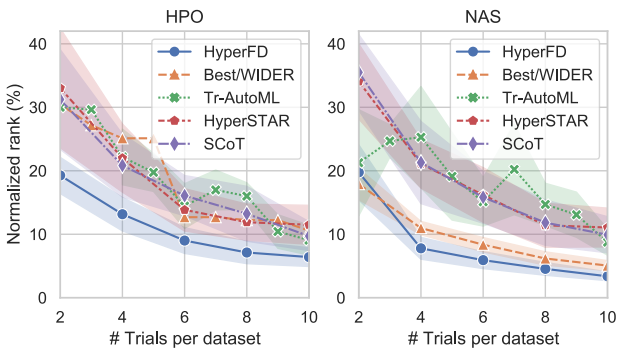

Figure 3. Performance (rank normalized to 0–100%) of HyperFD compared to baselines when assigned with different budgets. The shaded region is the standard deviation for 20 runs.

simple random search baseline. We think this could be attributed to three reasons. (i) The meta-feature they proposed to measure the similarity of datasets does not align with the face detection scenario. (ii) The loss functions they used (*e.g.*, MSE) make the optimization difficult. (iii) For Tr-AutoML, the design of sharing the "top-1" configuration among datasets is too arbitrary and does not sufficiently exploit the budget. To verify our hypothesis, we create variants of HyperFD by replacing the meta-feature extractor and loss function with those used in baselines. The results are shown in Table 1, which are significantly worse than HyperFD, indicating that the meta-feature extractor and loss function tailored for our scenario are the indispensable components to make the algorithm work. Among the three variants, the MSE version has the worst performance, implying that ranking loss is the most crucial to our results.

HyperFD remarkably outperforms baselines on all metrics, especially in terms of "rank". It is noteworthy that the improvement is particularly significant on the HPO search space. By contrast, on the NAS search space, most of the algorithms enjoy a better rank. We argue that NAS is an easier search space for AutoML, mainly because the datasets share more preferences for architectures than for hyperparameters. The findings in § 4.4 also echo such claim.

Finally, we verify the effectiveness of HyperFD by varying the budget, *i.e.*, number of trials to run for each dataset, between 2 and 10. We then report the averaged rank for each method under different budgets. The results are shown in Figure 3. Remarkably, HyperFD almost consistently performs best under all variants of budgets.

4.3. Ablations

Continual learning. We first evaluate the effectiveness of the proposed continual learning strategy. Specifically, we conduct several ablation experiments including: (i) Only leverage the tuning experience from the latest dataset. (ii) Freeze the meat-feature extractor after warm-up. (iii) Freeze the whole performance ranker after warm-up. The results are in Table 2. Thanks to the design of our performance ranker and the warm-up process, even if the whole ranker is frozen, the performance still looks fairly good. Nevertheless, using continual learning techniques can make the framework perform even better.

Method	HPO		NAS	
	ΔAP (↑)	Rank (↓)	ΔAP (↑)	Rank (↓)
Latest data only	1.12	15.99	2.03	9.55
Freeze meta-extractor	1.22	15.28	2.19	9.41
Freeze whole ranker	1.32	16.85	2.26	8.60
HyperFD (full)	**1.67**	**13.16**	**2.39**	**7.78**

Table 2. Ablations to justify the necessity of continual learning.

Method	HPO		NAS	
	ΔAP (↑)	Rank (↓)	ΔAP (↑)	Rank (↓)
No transformation	1.28	15.65	2.16	9.67
Oracle	1.84	13.79	2.37	7.73
HyperFD (full)	1.67	13.16	2.39	7.78

Table 3. Ablations on transformation module.

Ranking	Triplet	SI	HPO		NAS	
			ΔAP (↑)	Rank (↓)	ΔAP (↑)	Rank (↓)
✓		✓	0.50	16.49	1.60	12.06
✓	✓		1.38	15.40	2.33	8.86
✓	✓	✓	**1.67**	**13.16**	**2.39**	**7.78**

Table 4. Ablations on loss functions.

Transformation module. The proposed transformation module is a critical component of HyperFD. Equipped with the transformation module, HyperFD can benefit from the historical experience without sacrificing any privacy, because the transformation module only needs abstract dataset-level meta-features rather than raw images. From the results in Table 3, we can learn that the performance drops significantly without the transformation module, which means our design enables a more effective usage of the historical experience and alleviates the knowledge forgetting. Furthermore, we also compare HyperFD with the oracle baseline which neglects the privacy concerns and gathers all data at the central server. Remarkably, our transformation obtains comparable performance to the oracle baseline, which again proves the effectiveness.

Loss functions. The effectiveness of our loss functions are demonstrated in Table 4. There are three components in our loss function: a ranking loss and two regularization losses (triplet loss and SI loss). The ranking loss is essential and cannot be disabled, thus we turn the other two regularization losses on and off for comparison. On average, the triplet loss and SI loss contributes 3.81% and 1.66% to the rank across different search spaces.

Designs of meta-feature extractor. Apart from variants of meta-feature extractors in Table 1, we further experiment with various versions, where we disable several key components in our meta-feature extractor. For a more comprehensive comparison, we further show the validation NDCG [20], *i.e.* a ranking metric used to assess the warm-up quality. Table 5 empirically proves the effectiveness of

Method	HPO		NAS	
	NDCG(val) (↑)	Rank (↓)	NDCG(val) (↑)	Rank (↓)
Anchor-level: w/o labels	0.930	14.13	0.895	11.13
Image-level: w/o GWAP	0.932	14.22	0.901	10.41
Image-level: w/o pyramid	0.927	16.92	0.898	12.98
Image-level: w/o statistics	0.922	15.42	0.899	11.92
Dataset-level: w/o attn.	0.919	15.09	0.886	12.09
HyperFD (full)	**0.933**	**13.16**	**0.903**	**7.78**

Table 5. Ablations on meta-feature extractor. **w/o labels**: do not attach the bounding boxes information to feature maps. **w/o GWAP**: use average pooling rather than GWAP. **w/o pyramid**: use features from the last stage only. **w/o statistics**: no dataset-level descriptive statistics attached to the feature of each image. **w/o attn.**: remove the self-attention encoder layer.

Method	HPO		NAS	
	ΔAP (↑)	Rank (↓)	ΔAP (↑)	Rank (↓)
w/o warm-up	0.50	16.28	1.38	12.62
w/o augmentation	0.96	17.55	1.43	15.23
HyperFD (full)	**1.67**	**13.16**	**2.39**	**7.78**

Table 6. Ablations for warm-up and augmentation.

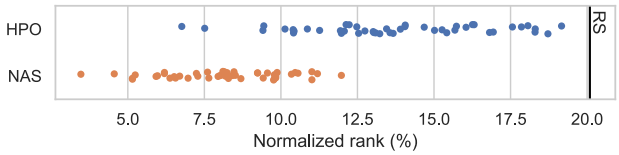

Figure 4. Distribution of normalized ranks of 50 experiments. RS: Random search.

our meta-feature extractor. Of the components in our extractor, the ablation of self-attention has the largest effect, with the worst NDCG(val) on both search spaces.

Warm-up and augmentation. Finally, we ablated our warm-up that is performed before the algorithm receives online datasets. We compare our method with the version without warm-up at all, and the version with warm-up but no augmented datasets. Results can be found in Table 6, where we see a performance drop without warm-up or augmentation. We argue that without proper warm-up, both meta-feature extractor and configuration encoder are prone to overfit the few samples (*i.e.*, trials) on each task. Moreover, meta-feature transformation module can hardly learn anything useful with only WIDER-Face dataset in hand.

4.4. Analysis

Robustness to task arriving order. The proposed HyperFD is robust to the order in which the tasks arrive. In Figure 4, we run 50 experiments with different arriving order for each search space, and report their distribution. Even the worst case is still much better than the average case of random search. Such robustness to order ensures the fairness across different tasks [65], which is important for a reliable platform.

Meta-feature visualization. We visualize representation of meta-features via t-SNE [49]. For each dataset, we extract meta-feature 30 times, each of which is extracted from a randomly sampled batch of images from the dataset. Results are shown in Figure 5. After warmed up, our meta-feature extractor can already successfully distinguish most of the datasets without further training the extractor on those datasets. There are two interesting findings. First, the datasets which are similar to each other (*e.g.*, WIDER-Face and WIDER-360) are also located close to each other in the figure. Second, the datasets tend to be more mingled among each other on NAS search space than on HPO search space, which means the preference on architectures is easier to be transferred among datasets than the preference on hyper-parameters. This also justifies the necessity to optimize the meta-feature extractor in an end-to-end manner.

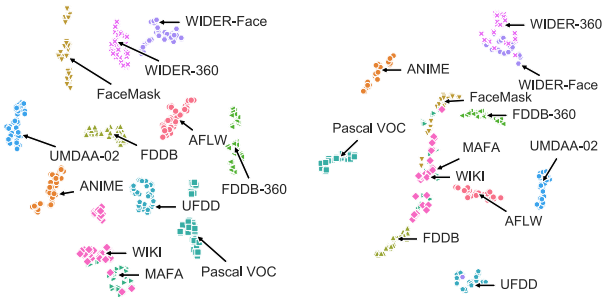

Figure 5. t-SNE visualization of the feature representations after offline warm-up. Each color represents a dataset. Multiple points correspond to multiple batches of images. **(left)** Trained on HPO search space. **(right)** Trained on NAS search space. (Better viewed in color)

5. Discussions and conclusions

From a platform perspective, this paper studied online AutoML for domain-specific face detection, *i.e.* how to continuously improve the AutoML algorithm and learn from a sequence of training tasks while protecting the privacy of sensitive face detection data. Various techniques are proposed and extensive experiments show their effectiveness. Admittedly, this paper did not include incorporation with multi-fidelity techniques (*e.g.*, BOHB [9]), and did not attempt to generalize our approach to scenarios other than face detection. We leave them in future works.

Broader impacts. The sensitivity of face datasets has been long noticed, and there is no need to emphasize more the importance of protect their privacy. However, if customers use their own datasets isolatedly, such over-protection will put resources in waste, especially when AutoML is used. By introducing HyperFD, we reach a balance between efficiency and privacy, and achieve the best of both worlds.

Acknowledgements

We thank anonymous reviewers for their valuable feedbacks. We also thank Chengmin Chi (STCA) and Xiaotian Gao (MSRA) for their suggestions.

References

[1] Alessandro Achille, Michael Lam, Rahul Tewari, Avinash Ravichandran, Subhransu Maji, Charless C Fowlkes, Stefano Soatto, and Pietro Perona. Task2vec: Task embedding for meta-learning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 6430–6439, 2019. 2

[2] Rémi Bardenet, Mátyás Brendel, Balázs Kégl, and Michele Sebag. Collaborative hyperparameter tuning. In *International conference on machine learning*, pages 199–207. PMLR, 2013. 2, 6

[3] Han Cai, Ligeng Zhu, and Song Han. Proxylessnas: Direct neural architecture search on target task and hardware. In *International Conference on Learning Representations*, 2018. 6

[4] Kai Chen, Jiaqi Wang, Jiangmiao Pang, Yuhang Cao, Yu Xiong, Xiaoxiao Li, Shuyang Sun, Wansen Feng, Ziwei Liu, Jiarui Xu, Zheng Zhang, Dazhi Cheng, Chenchen Zhu, Tianheng Cheng, Qijie Zhao, Buyu Li, Xin Lu, Rui Zhu, Yue Wu, Jifeng Dai, Jingdong Wang, Jianping Shi, Wanli Ouyang, Chen Change Loy, and Dahua Lin. Mmdetection: Open mmlab detection toolbox and benchmark, 2019. 4

[5] Yukang Chen, Tong Yang, Xiangyu Zhang, Gaofeng Meng, Xinyu Xiao, and Jian Sun. Detnas: Backbone search for object detection, 2019. 6

[6] Jiankang Deng, Jia Guo, Yuxiang Zhou, Jinke Yu, Irene Kotsia, and Stefanos Zafeiriou. Retinaface: Single-stage dense face localisation in the wild, 2019. 1, 3, 5

[7] Łukasz Dudziak, Thomas Chau, Mohamed S Abdelfattah, Royson Lee, Hyeji Kim, and Nicholas D Lane. Brp-nas: Prediction-based nas using gcns. *arXiv preprint arXiv:2007.08668*, 2020. 4, 6

[8] M. Everingham, S. M. A. Eslami, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The pascal visual object classes challenge: A retrospective. *International Journal of Computer Vision*, 111(1):98–136, Jan. 2015. 6

[9] Stefan Falkner, Aaron Klein, and Frank Hutter. Bohb: Robust and efficient hyperparameter optimization at scale. *arXiv preprint arXiv:1807.01774*, 2018. 8

[10] Jiemin Fang, Yuzhu Sun, Kangjian Peng, Qian Zhang, Yuan Li, Wenyu Liu, and Xinggang Wang. Fast neural network adaptation via parameter remapping and architecture search. In *International Conference on Learning Representations*, 2019. 5

[11] Matthias Feurer, Katharina Eggensperger, Stefan Falkner, Marius Lindauer, and Frank Hutter. Auto-sklearn 2.0: The next generation, 2020. 2

[12] Matthias Feurer, Jost Springenberg, and Frank Hutter. Initializing bayesian hyperparameter optimization via meta-learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 29, 2015. 2

[13] Jianglin Fu, Saeed Ranjbar Alvar, Ivan V. Bajic, and Rodney G. Vaughan. Fddb-360: Face detection in 360-degree fisheye images, 2019. 1, 5

[14] Jianglin Fu, Ivan V Bajić, and Rodney G Vaughan. Datasets for face and object detection in fisheye images. *Data in brief*, 27:104752, 2019. 6

[15] Nicolo Fusi, Rishit Sheth, and Melih Elibol. Probabilistic matrix factorization for automated machine learning. *Advances in neural information processing systems*, 31:3348–3357, 2018. 2

[16] Shiming Ge, Jia Li, Qiting Ye, and Zhao Luo. Detecting masked faces in the wild with lle-cnns. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017. 5

[17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015. 6

[18] Wobot Intelligence. Face mask detection dataset. https://www.kaggle.com/wobotintelligence/face-mask-detection-dataset, 2020. 5

[19] Vidit Jain and Erik Learned-Miller. Fddb: A benchmark for face detection in unconstrained settings. Technical Report UM-CS-2010-009, University of Massachusetts, Amherst, 2010. 5

[20] Kalervo Järvelin and Jaana Kekäläinen. Cumulated gain-based evaluation of ir techniques. *ACM Transactions on Information Systems (TOIS)*, 20(4):422–446, 2002. 5, 7

[21] Mingjie Jiang and Xinqi Fan. Retinamask: a face mask detector. *arXiv preprint arXiv:2005.03950*, 2020. 1

[22] Ziyue Jiang, Yi Ren, Ming Lei, and Zhou Zhao. Fedspeech: Federated text-to-speech with continual learning. *arXiv preprint arXiv:2110.07216*, 2021. 2

[23] Hadi S Jomaa, Lars Schmidt-Thieme, and Josif Grabocka. Dataset2vec: Learning dataset meta-features. *Data Mining and Knowledge Discovery*, 35(3):964–985, 2021. 2, 5

[24] Alexandros Kalousis. *Algorithm selection via meta-learning*. PhD thesis, University of Geneva, 2002. 2

[25] Jungtaek Kim, Saehoon Kim, and Seungjin Choi. Learning to warm-start bayesian hyperparameter optimization. *arXiv preprint arXiv:1710.06219*, 2017. 2, 3

[26] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017. 2

[27] Zhizhong Li and Derek Hoiem. Learning without forgetting. *IEEE transactions on pattern analysis and machine intelligence*, 40(12):2935–2947, 2017. 2

[28] Shengcai Liao, Anil K Jain, and Stan Z Li. A fast and accurate unconstrained face detector. *IEEE transactions on pattern analysis and machine intelligence*, 38(2):211–223, 2015. 1

[29] Linzaer. Ultra-light-fast-generic-face-detector-1mb. https://github.com/Linzaer/Ultra-Light-Fast-Generic-Face-Detector-1MB, 2020. 5

[30] Upal Mahbub, Sayantan Sarkar, and Rama Chellappa. Partial face detection in the mobile domain, 2017. 1, 6

[31] Peter M. Roth Martin Koestinger, Paul Wohlhart and Horst Bischof. Annotated Facial Landmarks in the Wild: A Large-scale, Real-world Database for Facial Landmark Localization. In *Proc. First IEEE International Workshop on Benchmarking Facial Image Analysis Technologies*, 2011. 5

[32] H. Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *AISTATS*, 2017. 2

[33] Donald Michie, David J Spiegelhalter, and Charles C Taylor. Machine learning, neural and statistical classification. 1994. 2

[34] Microsoft. Microsoft custom vision. 1

[35] Microsoft. Neural network intelligence. GitHub, 2018. 1

[36] Gaurav Mittal, Chang Liu, Nikolaos Karianakis, Victor Fragoso, Mei Chen, and Yun Fu. Hyperstar: Task-aware hyperparameters for deep networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8736–8745, 2020. 2, 3, 5, 6

[37] Hajime Nada, Vishwanath A Sindagi, He Zhang, and Vishal M Patel. Pushing the limits of unconstrained face detection: a challenge dataset and baseline results. In *2018 IEEE 9th International Conference on Biometrics Theory, Applications and Systems (BTAS)*, pages 1–10. IEEE, 2018. 6

[38] Mahyar Najibi, Pouya Samangouei, Rama Chellappa, and Larry Davis. Ssh: Single stage headless face detector, 2017. 3

[39] qhgz2013. anime-face-detector. https://github.com/qhgz2013/anime-face-detector, 2020. 5

[40] Suo Qiu. Global weighted average pooling bridges pixel-level localization and image-level classification, 2018. 4

[41] Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H Lampert. icarl: Incremental classifier and representation learning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 2001–2010, 2017. 2

[42] Rasmus Rothe, Radu Timofte, and Luc Van Gool. Dex: Deep expectation of apparent age from a single image. In *IEEE International Conference on Computer Vision Workshops (ICCVW)*, December 2015. 6

[43] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks, 2019. 5

[44] Matthew Schultz and Thorsten Joachims. Learning a distance metric from relative comparisons. *Advances in neural information processing systems*, 16:41–48, 2004. 5

[45] Hanul Shin, Jung Kwon Lee, Jaehong Kim, and Jiwon Kim. Continual learning with deep generative replay. *arXiv preprint arXiv:1705.08690*, 2017. 2

[46] Julien Siems, Lucas Zimmer, Arber Zela, Jovita Lukasik, Margret Keuper, and Frank Hutter. Nas-bench-301 and the case for surrogate benchmarks for neural architecture search. *arXiv preprint arXiv:2008.09777*, 2020. 4

[47] Kevin Swersky, Jasper Snoek, and Ryan Prescott Adams. Multi-task bayesian optimization. 2013. 2

[48] Gido M Van de Ven and Andreas S Tolias. Three scenarios for continual learning. *arXiv preprint arXiv:1904.07734*, 2019. 2

[49] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(86):2579–2605, 2008. 8

[50] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017. 4

[51] Noranart Vesdapunt and Baoyuan Wang. Crface: Confidence ranker for model-agnostic face detection refinement, 2021. 1

[52] Wei Wen, Hanxiao Liu, Yiran Chen, Hai Li, Gabriel Bender, and Pieter-Jan Kindermans. Neural predictor for neural architecture search. In *European Conference on Computer Vision*, pages 660–676. Springer, 2020. 4

[53] Catherine Wong, Neil Houlsby, Yifeng Lu, and Andrea Gesmundo. Transfer learning with neural automl. *arXiv preprint arXiv:1803.02780*, 2018. 2

[54] Yuxin Xiao, Eric P Xing, and Willie Neiswanger. Amortized auto-tuning: Cost-efficient transfer optimization for hyperparameter recommendation. *arXiv preprint arXiv:2106.09179*, 2021. 2

[55] Hang Xu, Ning Kang, Gengwei Zhang, Chuanlong Xie, Xiaodan Liang, and Zhenguo Li. Nasoa: Towards faster task-oriented online fine-tuning with a zoo of models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5097–5106, 2021. 2, 6

[56] Ju Xu and Zhanxing Zhu. Reinforced continual learning. *arXiv preprint arXiv:1805.12369*, 2018. 2

[57] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018. 4

[58] Yuanyuan Xu, Wan Yan, Haixin Sun, Genke Yang, and Jiliang Luo. Centerface: Joint face detection and alignment using face as point. In *arXiv:1911.03599*, 2019. 5

[59] Chao Xue, Junchi Yan, Rong Yan, Stephen M Chu, Yonggang Hu, and Yonghua Lin. Transferable automl by model sharing over grouped datasets. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9002–9011, 2019. 2, 6

[60] Chengrun Yang, Yuji Akimoto, Dae Won Kim, and Madeleine Udell. Oboe: Collaborative filtering for automl model selection. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1173–1183, 2019. 2

[61] Shuo Yang, Ping Luo, Chen Change Loy, and Xiaoou Tang. Wider face: A face detection benchmark. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. 5, 6

[62] Dani Yogatama and Gideon Mann. Efficient Transfer Learning Method for Automatic Hyperparameter Tuning. In Samuel Kaski and Jukka Corander, editors, *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics*, volume 33 of *Proceedings of Machine Learning Research*, pages 1077–1085, Reykjavik, Iceland, 22–25 Apr 2014. PMLR. 2, 4

[63] Jaehong Yoon, Wonyong Jeong, Giwoong Lee, Eunho Yang, and Sung Ju Hwang. Federated continual learning with weighted inter-client transfer. In *International Conference on Machine Learning*, pages 12073–12086. PMLR, 2021. 2

[64] Jaehong Yoon, Saehoon Kim, Eunho Yang, and Sung Ju Hwang. Scalable and order-robust continual learning

with additive parameter decomposition. *arXiv preprint arXiv:1902.09432*, 2019. 2

[65] Jaehong Yoon, Saehoon Kim, Eunho Yang, and Sung Ju Hwang. Scalable and order-robust continual learning with additive parameter decomposition, 2020. 8

[66] Jaehong Yoon, Eunho Yang, Jeongtae Lee, and Sung Ju Hwang. Lifelong learning with dynamically expandable networks. *arXiv preprint arXiv:1708.01547*, 2017. 2

[67] Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual learning through synaptic intelligence. In *International Conference on Machine Learning*, pages 3987–3995. PMLR, 2017. 2, 5

[68] Kaipeng Zhang, Zhanpeng Zhang, Zhifeng Li, and Yu Qiao. Joint face detection and alignment using multitask cascaded convolutional networks. *IEEE Signal Processing Letters*, 23(10):1499–1503, 2016. 1

[69] Shifeng Zhang, Xiangyu Zhu, Zhen Lei, Hailin Shi, Xiaobo Wang, and Stan Z. Li. Faceboxes: A cpu real-time face detector with high accuracy, 2018. 1

[70] Yuge Zhang, Chenqian Yan, Quanlu Zhang, Li Lyna Zhang, Yaming Yang, Xiaotian Gao, and Yuqing Yang. Acenas: Learning to rank ace neural architectures with weak supervision of weight sharing, 2021. 5