Submission and Formatting Instructions for International Conference on Machine Learning (ICML 2018)

Anonymous Authors[1]

Abstract

Abstract

1. Introduction

Machine learning (ML) based on artificial neural networks is a fast-growing field in the domain of computer science. Instead of manually tweaking the features used for classification or regression, neural networks are able to extract them automatically. They have shown state-of-the-art performance on a variety of classification problems when trained on huge datasets and are well suited for parallelization.

A open question is how to deal with computational limitations and generic datasets. In many real-world applications (e.g. smartphones) computational resources, execution time and storage capacity is limited. Thus the current trend of using larger and larger networks for better classification results becomes unfeasible. In addition, it may be not known a priori what the dataset to be classified looks like, requiring the neural network to be applicable to a variety of different datasets. Multiple approaches emerged within the last years to tackle both challenges:

To reduce the model size (and thus implicitly also execution time, storage capacity and computational resources), one can remove the most “unimportant” parts of a neural network and/or quantize it, resulting in a compression of the neural network The selection of the nodes/layers to be pruned can either be specified manually or derived automatically through reinforcement learning (Han et al., 2015; He et al., 2018). A second option is knowledge distillation where a smaller student“network” is trained to produce outputs similar to a previously trained larger “teacher” network (Phuong & Lampert, 2019; **?**). By using the logits output instead of the softmax output, much more information is preserved in every training step. Thus training the student network can be achieved with way less samples. For both approaches there is only a negligible decrease in classification accuracy while the compression rate can be huge (up to a factor of 496x for certain layers of a neural network (Reagan et al., 2018)). Another more straightforward method is to search directly for smaller/faster models. This can be either done manually (e.g. (Howard et al., 2017)) or through neural architecture search (Elsken et al., 2017). Note that the last approach can output networks that are state of the art with respect to model size, speed and classification accuracy (Tan & Le, 2019).

Other techniques are used if the time budget for training is tight or the dataset to be trained on is not known a priori. They all have in common that a network is pretrained on one or more datasets beforehand and information from pretraining is subsequently used for the final training step. One technique is transfer learning (Pan & Yang, 2010). Here the common use case consists of pretraining a network on a given dataset to initialize the network weights with reasonable values. If a new dataset is similar enough, only parts of the network must be trained again. In addition, the new dataset can be rather small as less optimization steps are required (the network is not trained from scratch). Another option is meta-learning or learning to learn (Hutter et al., 2018; Santoro et al., 2016): By learning multiple models for multiple tasks (or more specific: datasets) beforehand, it is quite likely that a new task will be similar to a previously learned task. In this case, knowledge gained from the previous task can be reused, thus accelerating the learning progress. The challenge with meta-learning is a proper selection of the meta-features to be extracted as their predictive capabilities vary heavily (Bilalli et al., 2017). One predominant meta-learning approach at the moment is MAML (Finn et al., 2017), optimizing a model not for classification accuracy but specifically for generalization performance.

The AutoDL challenge contains both aspects, tight computational limitations (both time- and resource-wise) and generic datasets. As such it provides an ideal testbed for advanced approaches in order to find data-savvy and time-efficient algorithms. Because the datasets are undisclosed, the options for manual tweaking are limited. Instead of highly specific solutions tied to a specific environment one

[1] Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

is thus seeking for a fast yet robust approach. This makes the challenge different from "traditional" benchmarks like ImageNet classification where computing power is solely limited by the budget of the chair/institute and the dataset is known beforehand.

The remainder of the paper is organized as follows: Sec. 2 describes the general problem from a more mathematical point of view. Sec. TODO provide general background information about the AutoDL challenge, the used datasets and initial approaches for the AutoDL video competition. As focus shifted after the video competition, Sec. TODO illustrates two more elaborate approaches for video and image classification that can be used for the final AutoDL challenge.

2. Problem formulation

3. AutoDL challenge

This section gives a quick overview of the AutoDL challenge: The AutoDL challenges consist of a set of different, individual challenges depending on the underlying dataset modality: Image (AutoCV), video (AutoCV2), text (AutoNLP), speech (AutoSpeech), weakly supervised learning (AutoWSL) and all previously mentioned modalities combined (AutoDL). All challenges have in common that teams can upload code and models during a online phase which is tested immediately on a number of datasets. This gives every team immediate feedback about their relative performance. These online phases last between two weeks and two months. Some challenges (e.g. for video) have a second, full blind testing (onsite) phase at the end where every team's last submission is evaluated on a new, previously unknown set of of datasets.

Differing from other challenges, there are tight constraints: Rather than focusing on the final classification accuracy after a certain time limit, each team can choose freely how much time to spend on training and how much on testing within a specific time budget $T = 1200s$. Training is performed by running a "train" function with the train dataset as input, test by running a "test" function with the test dataset as input and the classification results as output. Based on the output of every test run its score $s(t)$ at time t is calculated as

$$s(t) = 2 * AUC - 1,$$

with AUC as the area under receiver operating characteristic curve. The final score ALC is then calculated as the time integral over all individual scores $s(t)$, weighted on a logarithmic time scale as

$$ALC = \frac{1}{\log\left(1 + \frac{T}{t_0}\right)} \int_0^T \frac{s(t)}{t + t_0} dt,$$

with $t_0 = 20s$. This way, the predictions of every test run at time t are weighted with $\frac{1}{t+t_0}$. Because early predictions are weighted more, focus shifts from a high classification accuracy at the end of the time budget to a high classification accuracy as early as possible. A example of ALC can be seen in Fig. 1. It is visible that the performance within the first $60s$ counts roughly as much as the performance within the last $600s$. Additional limitations consider memory constraints (max. 300Mb per submission), computational constraints (NVIDIA Tesla P100 1GPU, 4vCPU, 26GB) and time constraints (AutoCV2: max. 5 hours runtime per 24 hours, max. 5 submissions per 24 hours).

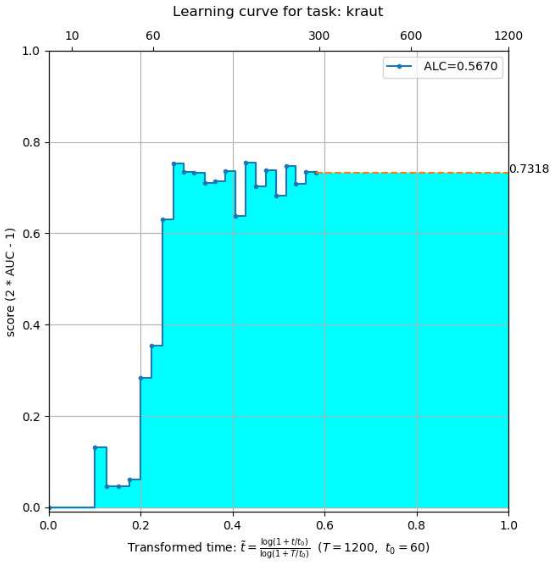

Figure 1. Visualization of the ALC plotted over time

4. Datasets

Table 1 lists all available datasets used for training our models. Shown is the dataset name both for image and video datasets, the number of train samples, test samples and classes and the source from where the dataset can be obtained. Datasets marked with "tfds" can be downloaded using the Tensorflow Datasets module, the ones marked with "AutoDL" using the provided code from the AutoDL challenge and the ones with "inet" are publicly available. Both the Youtube8m and YFCC100m have been downloaded partially by using/extracting the individual video links in/from the dataset file. Note that these two datasets are also the only two multilabel datasets. Although looking promising in theory as they are both the largest video datasets available and the only multilabel datasets, Youtube8m and YFCC100m were impossible to use due to their sheer size. Even if the exact reason is unknown (pos-

image dataset name	train samples	test samples	classes	source
Binary Alphadigits	1115	289	36	tfds
Caltech101	3059	6085	102	tfds
CUB-200	3000	3033	200	tfds
CUB-200-2011	5994	5794	200	tfds
Cats and Dogs	22263	999	2	tfds
CIFAR-10	50000	10000	10	tfds
CIFAR-100	50000	10000	100	tfds
COIL-100	6202	998	72	tfds
Colorectal Histology	4003	997	8	tfds
DeepWeeds	16520	989	5	tfds
EuroSAT	25976	1024	10	tfds
EMNIST	697932	116323	62	tfds
Fashion-MNIST	60000	10000	10	tfds
Horses or Humans	1027	256	2	tfds
KMNIST	60000	10000	10	tfds
MNIST	60000	10000	10	tfds
Oxford-102 Flower	1020	6149	102	tfds
Oxford-IIIT Pet	3680	3669	37	tfds
PatchCamelyon	262144	32768	2	tfds
Rock-Paper-Scissors	2520	372	3	tfds
THE small NORB	24300	24300	5	tfds
SVHN cropped	73257	26032	10	tfds
tf flowers	2938	732	5	tfds
UC Merced	1678	422	21	tfds
Chucky	48061	11939	100	AutoDL
Decal	634	166	11	AutoDL
Hammer	8050	1965	7	AutoDL
Munster	60000	10000	10	AutoDL
Pedro	80095	19905	26	AutoDL

video dataset name	train samples	test samples	classes	source
Katze	1528	863	6	AutoDL
Kreatur	1528	863	4	AutoDL
Kraut	1528	863	4	AutoDL
EPIC-Kitchens	24699	3773	125*331	inet
HMDB51	3570	500	51	inet
JHMDB21	658	270	21	inet
Kinetics 400	238831	19675	400	inet
something-something v2	168913	1036	174	inet
UCF101	9537	3783	101	inet
Youtube8m	1459266	216409	3862	own
YFCC100m	521035	130258	1570	own

Table 1. Dataset overview

sible inode overhead due to many files), training any neural network on them was almost a magnitude slower compared to smaller datasets. Furthermore, results from the AutoCV2 challenge indicate that it is unnecessary to capture multiple frames from a video (one is sufficient), thus turning the video classification task into a image classification task.

5. Image/video classification through transfer learning

The approach used for the AutoCV2 challenge is described in this section. Focusing solely on video classification, the AutoCV2 challenge consists of 5 publicly available datasets that can be downloaded during the online phase, 5 disclosed datasets used for the preliminary ranking during the online phase and another 5 disclosed datasets for the final evaluation during the on-site phase.

The general structure of most video classification networks is bipartite. Given a video V consisting of individual frames v_i as $V = [v_1, v_2, \ldots, v_n]$, a subset of frames $V' = [v'_1, v'_2, \ldots, v'_m] \subseteq V$ is selected. Typical selection choices are random, random within consecutive video segments or random with a bias towards the mid of the video. Every selected frame is then processed individually by a image classification network $f(\cdot)$ (e.g. a ResNet or InceptionNet) as $f(v'_i)$. The results of all processed frames - either on logits or feature level - is fed to a second network $g(\cdot)$ learning the temporal aspect between the frames. The classification result c is then calculated as

$$c = g\left(f(v'_1), f(v'_2), \ldots, f(v'_m)\right). \tag{1}$$

Both the 300Mb limit for each submission and the logarithmic time scale impose tight constraints. As such focus is on fast training rather than a superb classification rate. Hence MobileNet is chosen as primary network for the image classification f. Besides achieving state-of-the-art results on ImageNet, it can be arbitrarily scaled down for a better performance-accuracy tradeoff and is almost a magnitude smaller and faster than comparable networks. However, it is quite difficult to train. Therefore an Inception v3 network is added to the model portfolio.

dataset	avg. ALC score	rank (of 20)
dataset 1	0.1836	20
dataset 2	0.9138	2
dataset 3	0.4009	10
dataset 4	0.5169	1
dataset 5	0.1031	14

Table 2. AutoCV2 results

Different networks are evaluated for selecting the network g. In the end, a simple averaging layer shows sufficient performance while being fastest to train. Eq. (1) thus simplifies to

$$c = \frac{1}{m} \sum_{i=1}^{m} f(v'_i) \tag{2}$$

The averaged ALC scores based on three runs together with final ranks for the five datasets of the on-site phase are shown in Tab. 2. In total there are 20 participating teams. It is interesting to see how the ranks vary, from first place for dataset 4 to last place for dataset. We assume that dataset 4 is an average to difficult video classification dataset - this is what our network has been optimized for. Conversely the used network is rather slow compared to some deep learning unrelated approaches or very small networks. This causes problems if the test dataset is large or easy to classify.

6. Image/video classification through meta-learning

The AutoCV2 challenge has shown severe limitations of the transfer-learning approach using only two models, specifically low adaptability to changed conditions (e.g. very easy/difficult classification task or large test dataset). Thus different approaches based on meta-learning are presented in this chapter. They all have in common that a classifier (from now an called meta-classifier) is trained to measure the similarity between any new dataset and a set of known datasets using meta-features. Fur every known dataset an optimized neural network has been selected a priori. Then the neural network of the most similar known dataset will be used for transfer learning on the new dataset. However, they differ on the type of meta-features and the classifier that determines the similarity of the new dataset to the set of known datasets. The approaches belong to the larger class of non-parametric meta-learning and are conceptually similar with (Snell et al., 2017; Sung et al., 2018; Vinyals et al., 2016) in the sense that they first learn an embedding of the data and then measure the similarity on the embedding space. The difference is that we want to distinguish datasets and not samples of different classes within one dataset.

Formally, for all presented approaches the dataset to be learned is named task $t_j \in T$ and contained in the set of all known tasks. Each task is described by a meta-feature vector $\mathbf{m}(t_j)$. The neural network and its corresponding (hyper-)parameters are defined by a configuration $\theta_i \in \Theta$ where Θ represents the entire configuration space (discrete, continuous or mixed). In addition, there are offline evaluations $P(\theta_i, t_j) \in \mathbf{P}$ from the set of all prior evaluations $\mathbf{P}$. Assuming that we are given a new task (dataset) t_{new}, we want to specify how similar it is to prior (training) tasks t_j and derive an optimized configuration θ^*_{new} to maximize $P(\theta^*_{new}, t_{new})$.

It is assumed that an optimized configuration $\theta^*(t_j)$ is found for every task t_j during the offline phase. As all tasks are represented by their meta-feature vectors $\mathbf{m}(t_j)$, we want to find the most similar task t_{sim} for a new task t_{new} as

$$t_{sim} = \underset{t_j \in T}{\operatorname{argmax}} \left\| \mathbf{m}(t_{new}) - \mathbf{m}(t_j) \right\|, \tag{3}$$

and then use the configuration $\theta^*(t_{sim})$ for t_{new}.

Two challenges arise from Eq. (3):

- Finding expressive meta-features $\mathbf{m}(t_j)$ for every task
- Finding a reasonable distance measure $\|\cdot\|$

Expressive meta-features are essential for distinguishing different tasks (a.k.a. datasets): If different datasets result in similar meta-feature vectors they cannot be distinguished, regardless of the used distance measure. At the same time the variation of the meta-features shall be roughly proportional to the variation of the datasets: An intuitive counterexample is the use of a generic hash function as meta-feature, which leads to large variations even if two datasets are almost similar. The same applies for the distance measure: Having found good meta-features, almost identical datasets shall have only a small distance whereas dissimilar datasets shall result in a large distance.

The next sections describe how both challenges are tackled by different approaches.

6.1. Approach 1

The first class of presented meta-classifers relies on the output of an image classification network as meta-features. Being trained on a sufficiently large dataset with enough features/classes (corresponding to the output of the second last/last network layer), the distribution of the detected features/classes acts as a unique “fingerprint” for each dataset and constitutes the meta-feature vector $\mathbf{m}(t_j)$. The advantage of the method is the large number of possible meta-features, i.e. the number of features/classes of the original dataset ($> 1e3$) and the fully automated classification. In addition, the network generating the vector $\mathbf{m}(t_j)$ can be trained offline and then kept frozen. A possible disadvantage is the expressiveness of the meta-features if new datasets contain classes that are orthogonal to the trained ones.

Different classifiers are used to find a reasonable distance metric based on the output of the image classification network, see Fig. 2. By assigning an individual label to every dataset (note that every dataset represents a class), the distance measure is learned implicitly.

The first approach (which also serves as baseline for the experiments) relies on k-means clustering to specify the most similar dataset for every sample of the new dataset. It also uses majority voting for more reliable predictions based on multiple samples (i.e. batch sizes > 1).

The second approach is based upon a two-layer perceptron (MLP) with a Swish activation function and dropout. Similar to the baseline approach, majority voting in combination with large batch sizes is used to make predictions less susceptible to outliers.

The third approach uses the same two-layer perceptron as the second approach. However, instead of relying on majority voting, the output of the image classification network is first fed to a precalculation layer. This layer calculates different p-quantile moments and related measures (mean, median, standard deviation, variance, skewness, ...) which are then fed to the two-layer perceptron. Note that the moments are calculated across the batch size, i.e. when calculating m moments for a batch size b and f meta-features, the output of the precalculation layer is a vector with length $m \cdot f$.

A visual overview of the different approaches together with typical implementations is given in Fig. 2. Here, a ResNet18 is used as image classification network. It also shows the output dimensions of the different components for a batch size of 128, an input image size of 96x96 pixels with three channels, a 512-dimensional ResNet18 output (the second-last layer), the use of five modes (e.g. mean, standard deviation, variance, skewness, kurtosis) resulting in the 2560-dimensonal (512x5) output of the precalculation layer and 20 datasets to be distinguished.

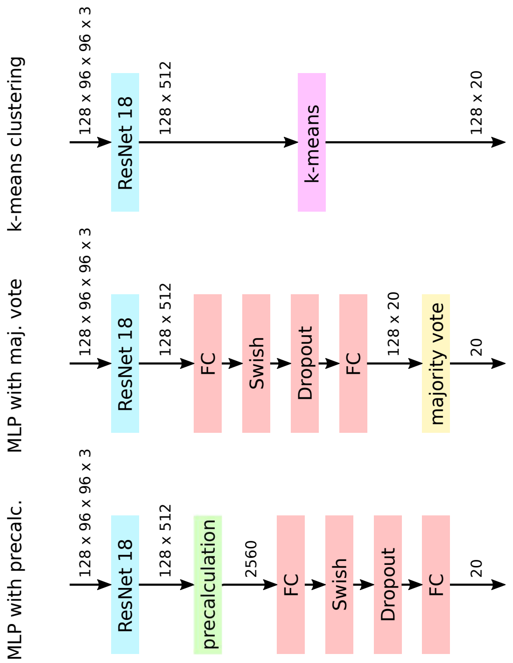

Figure 2. Overview of the different approaches based on an image classification network with typical implementation parameters

6.2. Approach 2

TODO: Philipp's approach

6.3. Experiments

BOHB is used for the a priori calculation of optimized configurations $\theta^*(t_j)$ for the different training datasets t_j. Every configuration consists of a pretrained network (either trained on ImageNet or CIFAR-100) and the following hyperparameters (if applicable):

- Optimizer: ADAM or SGD (with Nesterov momentum 0.9 and weight decay 1e-6)
- Train batch size: 16/32/64/128
- Learning rate: 1e-5 - 0.5
- Dropout 0.01 - 0.99

An overview of the used datasets and models is given in Fig. 3 and Fig. 4. Both figures show the best classification accuracy of the different models on the different datasets, both from a model-based perspective and a dataset-based perspective. The hyperparameters for every combination of dataset and model are optimized through 10 BOHB runs. Note that not all datasets listed in Tab. 1 appear in Fig. 3. Some are used as test datasets whereas others (especially video datasets) are too large to be processed efficiently with the given resources. In addition, multiple optimization runs find an optimal meta-feature vector and a reasonable distance measure while keeping the configuration space Θ small: Being reasonably small and fast, a ResNet18 network pretrained on ImageNet serves as basis to generate the meta-feature vector $\mathbf{m}(t_j)$. However, using the output of the second last layer yields consistently better results than the last output and is therefore chosen.

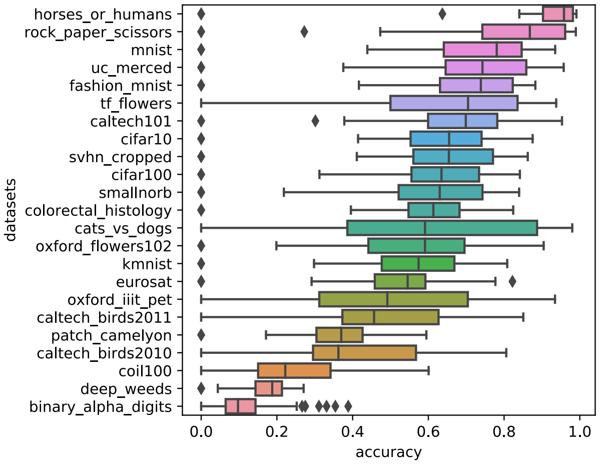

Figure 3. Classification accuracy of the different models across all datasets, sorted by dataset

The following parameters have been optimized for the different approaches using BOHB:

First approach (k-means):

- Number of clusters: 5 - 40
- Number of different initializations: 10 - 30
- Maximum number of iterations per run: 100 - 500

Second approach (MLP with majority vote):

- Train batch size: 1/2/4/8/16/32/64/128/256/512/1024
- Neurons per layer: 16/32/64/128/256/512/1024
- Learning rate: 1e-5 - 1e-2
- Dropout rate: 0 - 0.8

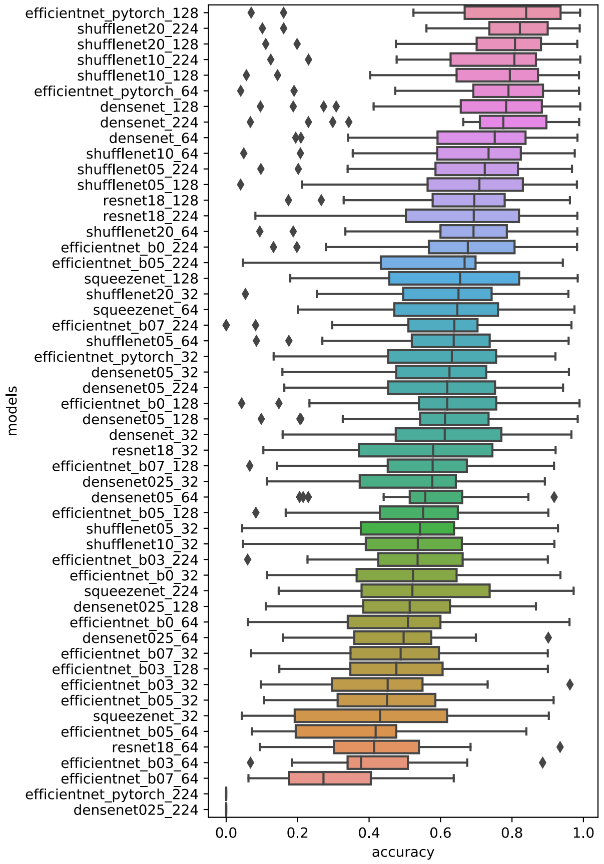

Figure 4. Classification accuracy of the different models across all datasets, sorted by model

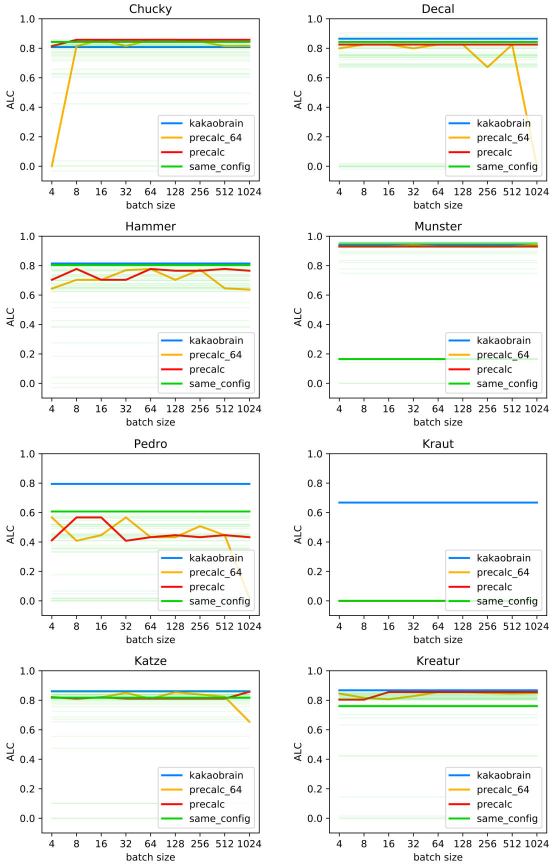

Figure 5. Final classification results

Third approach (MLP with precalculation):

- Train batch size: 1/2/4/8/16/32/64/128/256/512/1024
- Neurons per layer: 16/32/64/128/256/512/1024
- Learning rate: 1e-5 - 1e-2
- Dropout rate: 0 - 0.8
- p-quantile: 0 - 0.4
- (Do not) use mean, median, standard deviation, variance, skewness, kurtosis

Final results of the different approaches are shown in Fig. 5. The eight plots correspond to the eight test datasets (Chucky, Decal, Hammer, Munster, Pedro, Kraut, Katze, Kreatur). Every plot shows the test ALC according to Eq. (1) over the test batch size of the classification networks of chapter 6.1 and 6.2. The different thick solid lines correspond to:

- **kakaobrain**: ALC results from the winning AutoCV and AutoCV2 team kakaobrain (**?**).
- **same-config**: ALC of the optimized configuration $\theta^*(t_s)$ for the same dataset t_s. Can be seen as ground truth.
- **precalc**: Third approach of chapter 6.1 with individual networks trained specifically for every test batch size.
- **precalc-64**: Third approach of chapter 6.1 with one network trained for a test batch size of 64 samples.

There are also multiple horizontal thin green lines in every plot. Each one corresponds to the ALC of the optimized configuration $\theta^*(t_p)$ of a prior dataset t_p, evaluated on the test dataset. Thus, a plot ideally has the following appearance: Kakaobrain has the highest ALC as we do not expect to perform better than a team of computer vision professionals. Next comes the ALC of the optimized configuration for the corresponding dataset (same-config). Note that this measure is usually not available and only added for comparative purpose. The individual ALCs of the different presented approaches are expected to score worse than the previous two as they rely on optimized configurations of different datasets (and not the test dataset).

There are several effects why the plots in Fig. 5 differ from the ideal plot described before:

Sometimes an optimized configuration has a better ALC than kakaobrain. A possible reason is the different ALC calculation: Due to heavily varying execution times on the cluster (up to a factor of 10!) we did not measure the real ALC. Rather, the execution time of an individual forward and backward pass for every model for images/videos of different input shapes has been measured on a reference system and been stored in a lookup table. The lookup table is then used on the cluster to estimate the ALC based on the number of batches, the number of samples per batch and the average shape of the images/videos of the dataset. This procedure allows one to be independent of the cluster execution time while having a good approximation of the true ALC but may overestimate the true ALC.

In addition, the same-config ALC is sometimes worse than the ALCs of optimized configurations of prior datasets. We are not sure about the exact cause. One reason can be the limited number of BOHB evaluations to find the optimized configuration $\theta^*(t_s)$ for the same dataset. Only 90 different configurations (10 BOHB runs, eta=3, max budget = 9 * min budget) are evaluated for every dataset. Thus, with more than 20 other training datasets used, it is possible that one of these 20 optimized configurations scores better. Another option is the ALC variation during test time. Hence an optimized configuration which scored rather well during training may score considerably worse during testing (even on the same dataset).

For some datasets (e.g. Munster) the ALC of a presented approach is similar for multiple/all test batch sizes. Although this is the preferred case (the output of the meta-classifier is ideally independent of the batch size), the effect is usually caused by the test dataset being similar to one of the training datasets. Then a well-trained meta-classifier always correctly classifies the test dataset being most similar to the (identical) training dataset.

Choosing an optimal test batch size is crucial during test time. When being too small, the meta classifier may produce noisy (and thus possibly incorrect) predictions. When being too large, more samples than necessary are processed. This leads to computational overhead. TODO: further explanation.

7. Discussion

TODO

The used HDDs of the computing cluster constituted a main bottleneck during calculation and led to a huge variance in execution speed - sometimes up to a factor of 10. In combination with a measure based on accuracy over time, reproducible results became virtually impossible. Thus the workaround with the lookup table was chosen.

8. Summary and Outlook

TODO

References

Bilalli, B., Abelló, A., and Aluja-Banet, T. On the predictive power of meta-features in openml. *Applied Mathematics and Computer Science*, 27:697–712, 2017.

Elsken, T., Metzen, J. H., and Hutter, F. Nips, 2017.

Finn, C., Abbeel, P., and Levine, S. Model-agnostic meta-learning for fast adaptation of deep networks. In *ICML*, 2017.

Han, S., Mao, H., and Dally, W. J. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. In *ICLR*, 2015.

He, Y., Lin, J., Liu, Z., Wang, H., Li, L., and Han, S. AMC: automl for model compression and acceleration on mobile devices. In *ECCV*, 2018.

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *ArXiv*, 2017.

Hutter, F., Kotthoff, L., and Vanschoren, J. (eds.). *Automatic Machine Learning: Methods, Systems, Challenges*. Springer, 2018.

Pan, S. J. and Yang, Q. A survey on transfer learning. *IEEE Trans. on Knowl. and Data Eng.*, 22(10):1345–1359, October 2010. ISSN 1041-4347. doi: 10.1109/TKDE.2009.191.

Phuong, M. and Lampert, C. Towards understanding knowledge distillation. In *ICML*, 2019.

Reagan, B., Gupta, U., Adolf, B., Mitzenmacher, M., Rush, A., Wei, G.-Y., and Brooks, D. Weightless: Lossy weight encoding for deep neural network compression. In *ICML*, 2018.

Santoro, A., Bartunov, S., Botvinick, M., Wierstra, D., and Lillicrap, T. Meta-learning with memory-augmented neural networks. In *ICML*, 2016.

Snell, J., Swersky, K., and Zemel, R. Prototypical networks for few-shot learning. In *NIPS*, 2017.

Sung, F., Yang, Y., Zhang, L., Xiang, T., Torr, P. H. S., and Hospedales, T. M. Learning to compare: Relation network for few-shot learning. In *CVPR*, 2018.

Tan, M. and Le, Q. EfficientNet: Rethinking model scaling for convolutional neural networks. In *ICML*, 2019.

Vinyals, O., Blundell, C., Lillicrap, T., Kavukcuoglu, K., and Wierstra, D. Matching networks for one shot learning. In *NIPS*, 2016.