REDUCING NEURAL ARCHITECTURE SEARCH SPACES WITH TRAINING-FREE STATISTICS AND COMPUTATIONAL GRAPH CLUSTERING

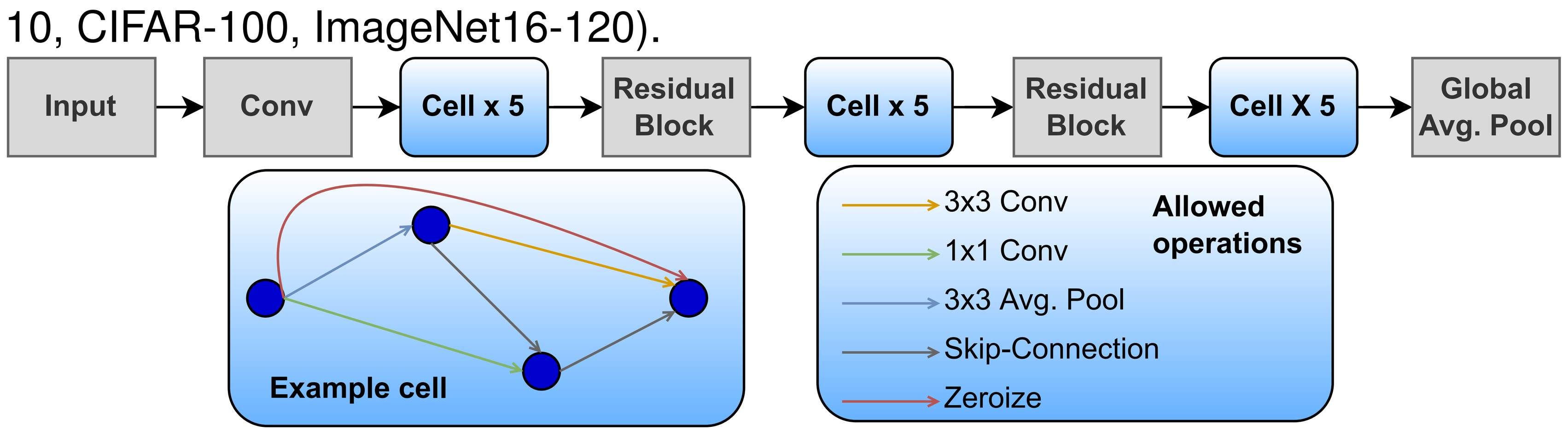
Thorir Mar Ingolfsson[1], Mark Vero[1], Xiaying Wang[1]
Lorenzo Lamberti,[2], Luca Benini[1,2], Matteo Spallanzani[1]
[1]ETH Zürich, [2]Università di Bologna

Why reducing NAS spaces?

Neural Architecture Search (NAS) aims at discovering Deep Neural Network (DNN) topologies that have good task accuracy. NAS algorithms are often time-consuming and computationally expensive; thus, being able to focus the search on those sub-spaces containing good candidates can greatly improve the performance of NAS algorithms. This work investigates how to efficiently identify high-performing subspaces of a given NAS space.

We model the **DNN architecture** as a latent variable $\lambda \in \Lambda$. This variable manifests itself through several *observable properties*, mainly 1) the **functional form** f_λ and 2) the **program form** G_λ.

We validate our ideas on the **NAS-Bench-201** (NB201) dataset. NAS-Bench networks are built by concatenating six identical *cells*, which can be configured in 5^6 different ways. NB201 contains $5^6 = 15,625$ DNNs, each of which is annotated with its task accuracy over three different image classification data sets (CIFAR-10, CIFAR-100, ImageNet16-120).



Training-Free Statistics

In its functional form, a DNN is a function

$$f_\lambda : \Theta_\lambda \times X \to Y$$

that is parametric in $\theta_\lambda \in \Theta_\lambda$. The parameter evolves from a randomly chosen initial condition $\theta_\lambda^{(0)} \sim \mu_{\theta_\lambda^{(0)}}$ over a stochastic trajectory $\{\theta_\lambda^{(0)}, \dots, \theta_\lambda^{(T)}\}$, where $T \in \mathbb{N}$ is the number of iterations of the training algorithm.

Let $D^n := (X \times Y)^n$ be the collection of all n-samples taken from $X \times Y$. We define D^0 to be the empty tuple, $D^+ := \cup_{n=1}^{+\infty} D^n$ to be the collection of non-empty samples, and $D^* := D^0 \cup D^+$ to be the collection of (possibly empty) samples.

Given a DNN f_λ, we define a **statistic** to be a measurable function

$$s_\lambda : \Theta_\lambda \times D^* \to S\,,$$

where S is some set of measurements. Note that we can measure the statistic for an untrained ($t = 0$), partially trained ($0 < t < T$), or completely trained ($t = T$) network parameter $\theta_\lambda^{(t)}$. Note also that the value of the statistic is stochastic in the point $\theta_\lambda^{(t)}$ of the parameter trajectory and in the sample $\mathcal{D} \in D^*$.

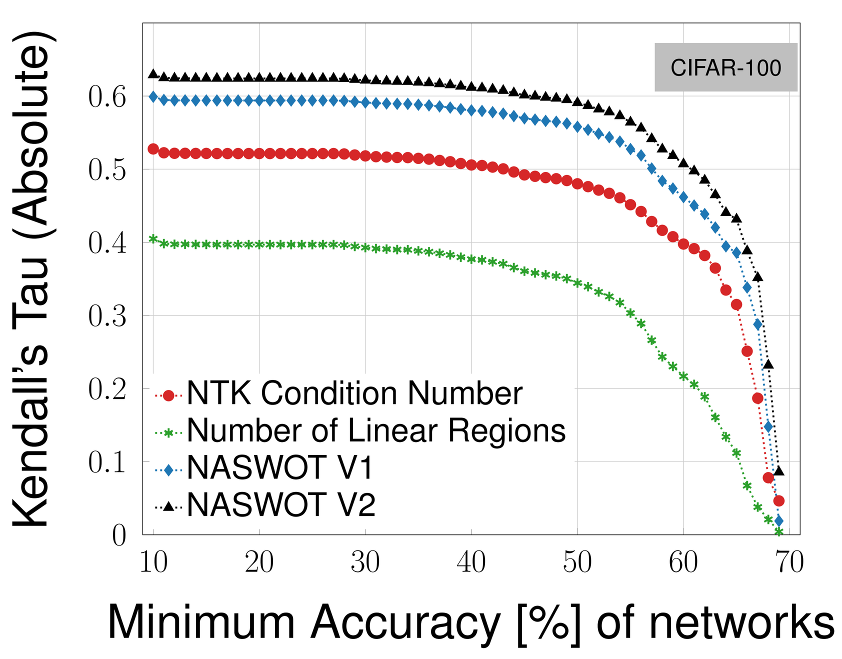
We can describe the latent architecture variable $\lambda \in \Lambda$ through the statistic s_λ after t training iterations by computing

$$\mathbb{E}_{\substack{\theta_\lambda \sim \mu_{\theta_\lambda^{(t)}} \\ \mathcal{D} \sim \mu_{D^*}}}[s_\lambda(\theta_\lambda, \mathcal{D})] \approx \frac{1}{N} \sum_{n=1}^{N} s_\lambda(\theta_\lambda^{(t,n)}, \mathcal{D}^{(n)})\,.$$

The most important statistic is **task accuracy**. The problem with task accuracy is that it must be computed with respect to $\mu_{\theta_\lambda^{(T)}}$, i.e., *it requires one to run the training algorithm to completion before it can be measured*. **Training-free (TF) statistics** are statistics whose distributions, when computed with respect to the distribution $\mu_{\theta_\lambda^{(0)}}$, correlates well with the distribution of task accuracy. *TF statistics provide a much cheaper estimate for the task accuracy of a candidate DNN, since they avoid the need to train it.*

NAS literature on TF statistics has so far proposed four candidate statistics:

- the condition number of the Neural Tangent Kernel (NTK);
- the number of regions cut in the input domain X by a ReLU-activated network;
- NAS w/o Training Score (version 1 and version 2).



Correlation between task accuracy and TF statistics hits sharp decline with progress towards high performing networks. TF statistics are therefore of limited use for pointwise information, but work great at **relational information**.

Clustering Computational Graphs

In its program form, a DNN is a bipartite graph

$$G_\lambda = (V_{M,\lambda}, V_{K,\lambda}, E_{R,\lambda} \cup E_{W,\lambda})$$

called a **computational graph (CG)**, where $V_{M,\lambda}$ is a set of memory nodes (e.g., parameters or feature arrays), $V_{K,\lambda}$ is a set of kernel nodes (e.g., convolutions or activation operations), $E_{R,\lambda} \subset V_{M,\lambda} \times V_{K,\lambda}$ is a set of read operations, and $E_{W,\lambda} \subset V_{K,\lambda} \times V_{M,\lambda}$ is a set of write operations. We can derive a CG composed only of arrays or operations by *projecting* it onto the memory or kernel partition, respectively.

We represent a computational graph G_λ using a third-order **adjacency tensor**

$$A_\lambda \in \{0,1\}^{|V_\lambda| \times |V_\lambda| \times |P|}\,,$$

where V_λ is the set of arrays building up G_λ, P is the collection of operation types, and $|\cdot|$ denotes set cardinality. We compare two graphs $G_{\lambda_1}, G_{\lambda_2}$ by using two classes of distances:

- probabilistic differences comparing the distributions of operations usage (symmetricised Kullback-Leibler, Jensen-Shannon, Hellinger);
- a transformation distance capturing the cost of transforming one graph into another; the cost model is defined heuristically.

Clustering algorithms (k-means, spectral clustering) create **groups of objects that are similar under the chosen distance**.

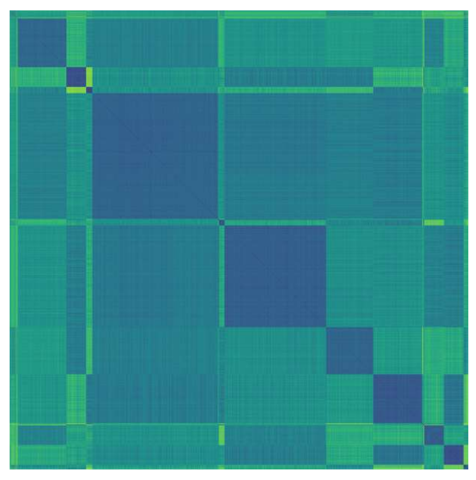
Algorithm 1 Clustering-Based REDuction (C-BRED)

Input: Architecture Search Space Λ, Distance Measure d, Clustering Algorithm $\mathcal{A}_\mathcal{C}$, Cluster Scoring Function f

Output: High-Quality Subspace $\Lambda^* \subseteq \Lambda$

```
1: K, η ← select_best_parameter(A_C, G_Λ, d)
2: {Λ_1, ..., Λ_K} ← A_C(G_Λ, d, K, η)
3: for i = 1 to K do
4:     f_i ← f(Λ_i)
5:     if (f_i ≥ best_score) then
6:         best_score ← f_i
7:         best_cluster ← Λ_i
8:     end if
9: end for
10: return Λ* ← best_cluster
```

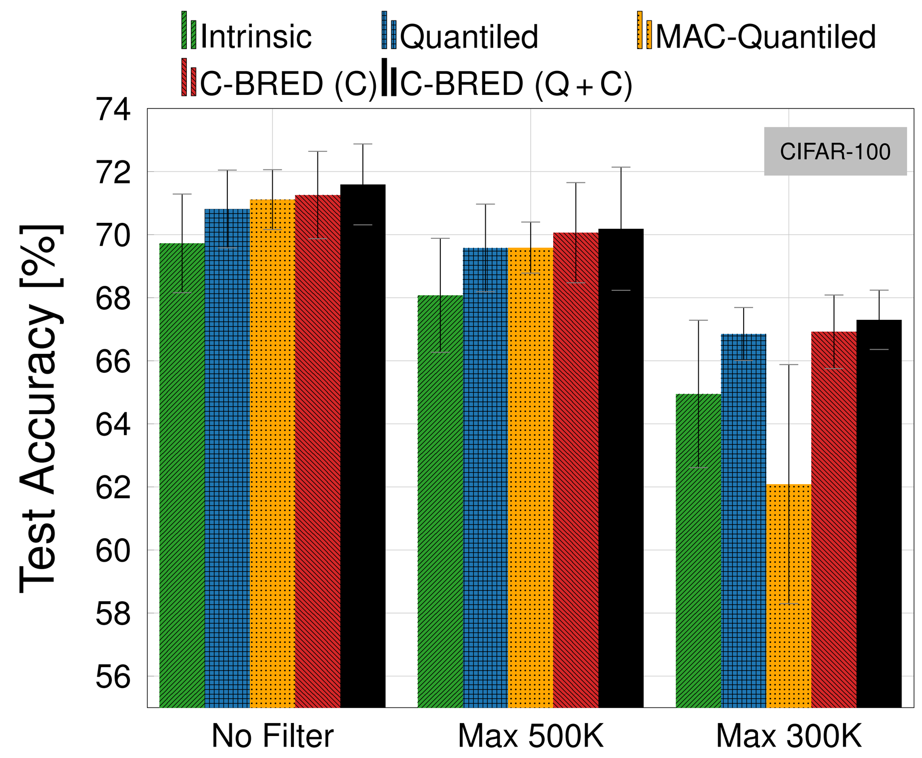
The similarity between programs also impacts the intra-cluster distributions of TF statistics: indeed, **different clusters exhibit different distributions of TF statistics**.



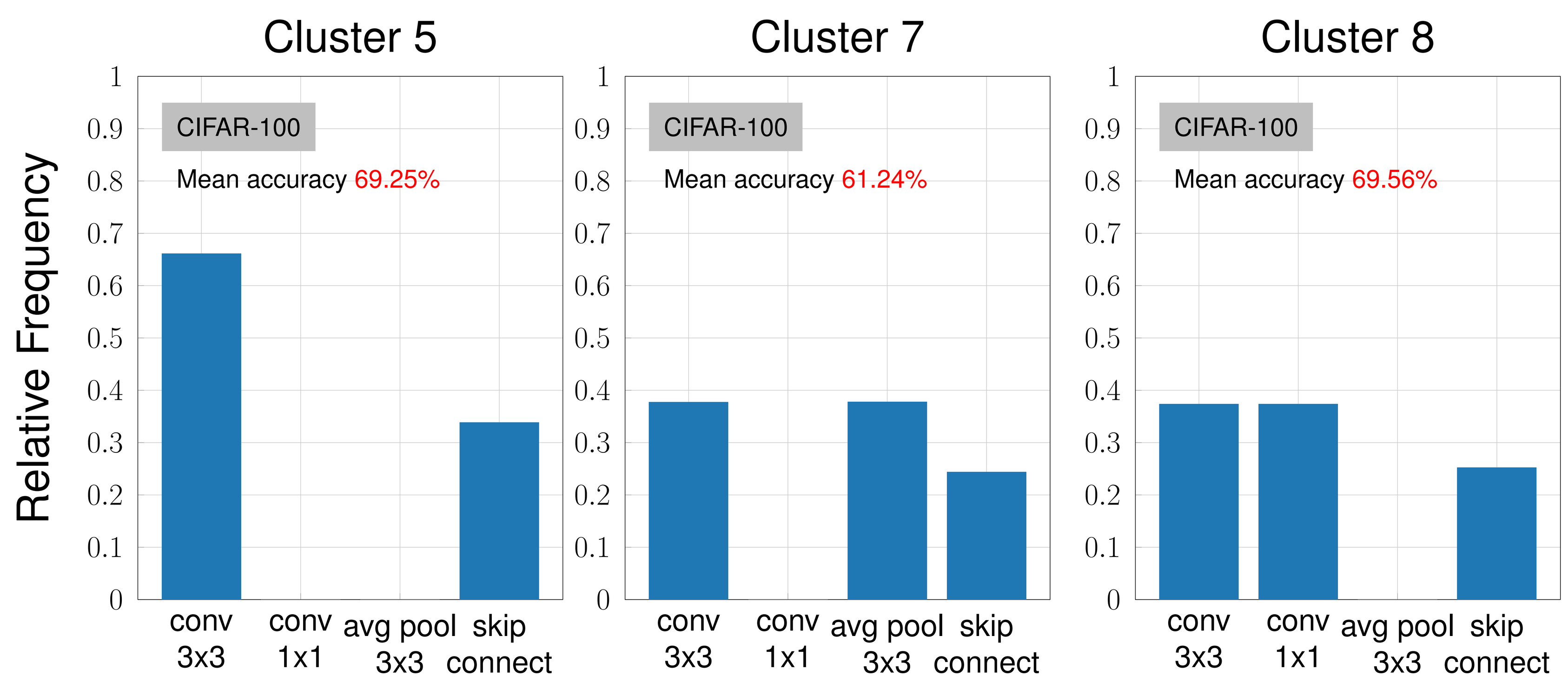
Experiments and Insights

We can exploit the observation of intra-cluster distributions of TF statistics to derive an **unsupervised algorithm to select those program clusters which have the best distributions of TF statistics**.

Even a trivial search algorithm such as sampling networks at random from the best cluster can find task-accurate networks.



Looking at the distributions of the operations used by the programs in the best clusters, we see 1) that **good programs do not use disruptive operations** (e.g., pooling) and 2) that **more compact programs mix 3×3 and 1×1 convolutions instead of using only 3×3 convolutions**. This can be seen from the figure below, but clusters 5 and 8 vastly outperform cluster 7.



Email: thoriri@iis.ee.ethz.ch