
2021
ICCV
OCTOBER 11-17
VIRTUAL

2021 ICCV OCTOBER 11-17 VIRTUAL

Fast and Efficient DNN Deployment via Deep Gaussian Transfer Learning

Qi Sun[1], Chen Bai[1], Tinghuan Chen[1], Hao Geng[1], Xinyun Zhang[2], Yang Bai[1], Bei Yu[1]

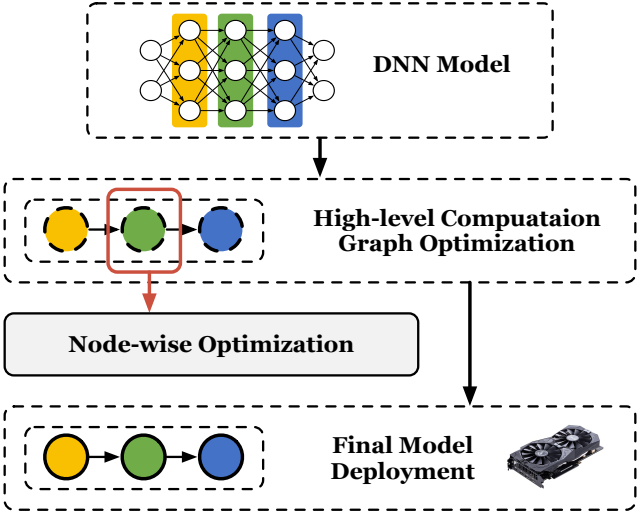
[1]The Chinese University of Hong Kong
[2]SmartMore
{qsun,byu}@cse.cuhk.edu.hk

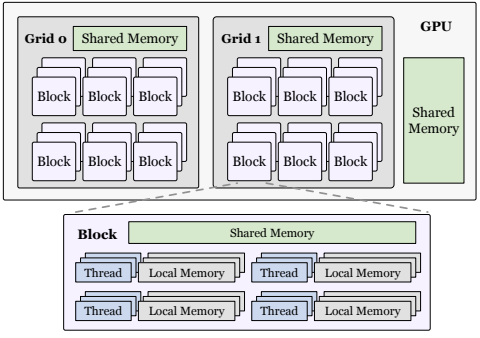
Oct. 15, 2021

SmartMore

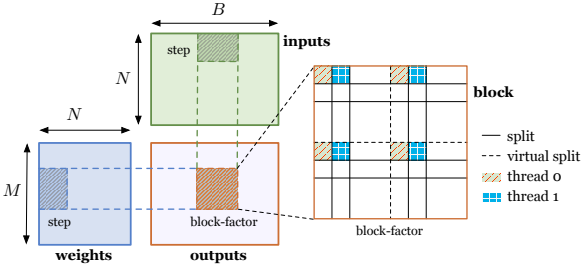
Background

Global View of DNN Deployment Framework





GPU Programming Model.



Map the computations to the programming model.

Deployment Configuration

All of the settings (*e.g.*, blocks, threads, and *etc.*) to be determined are encoded as a feature vector x which is termed a *deployment configuration*.

Design Space

For each DNN layer, the design space $\mathcal{D}$ contains all of the candidate configurations.

Typically, there are more than millions of candidates for each layer.

Optimization Objective

For each layer, find the deployment configuration $x_* \in \mathcal{D}$ which has the best performance.

Challenges

- Extremely large design space
- Slow compilation process
- Underutilized historical information

Our Solution

Deep Gaussian Transfer Learning

Transfer Learning Based on Deep Gaussian Processes

- Layer-wise optimization
- Stage 1 preparation: learn a deep Gaussian process model from historical data
  - model pre-training
- Stage 2 transfer: transfer knowledge of the DGP model to new tasks
  - model tuning
- Stage 3 optimal searching: guide the optimization of new tasks with the tuned DGP model
  - the tuned DGP model is used as the cost model in the searching algorithm

Deep Gaussian Processes with Stochastic Variational Inference

- task: $f : \boldsymbol{x} \to y$
- historical feature vectors $\boldsymbol{X} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$, objective GFLOPS values $\boldsymbol{y} = \{y_1, \ldots . y_N\}$

Deep Gaussian process: Multi-layer GP Model

- function values of L layers: $\{\boldsymbol{f}^1, \ldots, \boldsymbol{f}^L\}, \boldsymbol{f}^0 = \boldsymbol{X}$
- hyper-parameters in the l-th layer: $\boldsymbol{\theta}^l$
- sparse Gaussian approximations
- train the model through maximum log likelihood estimation:

$$\max_{\{\boldsymbol{z}^l, \boldsymbol{\theta}^l\}_{l=1}^L} \left[\sum_{i=1}^{N} \mathbb{E}_{\mathcal{Q}(f_i^L | \boldsymbol{u}; \boldsymbol{x}_i, \boldsymbol{Z})} [\log \mathcal{P}(y_i | f_i^L; \boldsymbol{\theta}^L)] \right] - \sum_{l=1}^{L} \mathcal{KL}[\mathcal{Q}(\boldsymbol{u}^l) \| \mathcal{P}(\boldsymbol{u}^l; \boldsymbol{\theta}^l)], \tag{1}$$

Transfer Knowledge to New Tasks

- Historical data: source task
- New deployment task: target task

Similarities and Differences

- Similar types of layers and hardware architectures.
- Different task hyper-parameters and different amounts of resources.

Two Steps

- Find a good tuning data set (from the target data set)
- Transfer the knowledge efficiently

Find A Good Tuning Set

- Randomly sample an initial set of configurations (target task)
- Use the learned DGP model (source task) to predict performance values
- Choose the configurations with top predicted values, denoted as $\boldsymbol{X}^t$
- Compile and deploy the configurations to get the real performance values $\boldsymbol{y}^t$

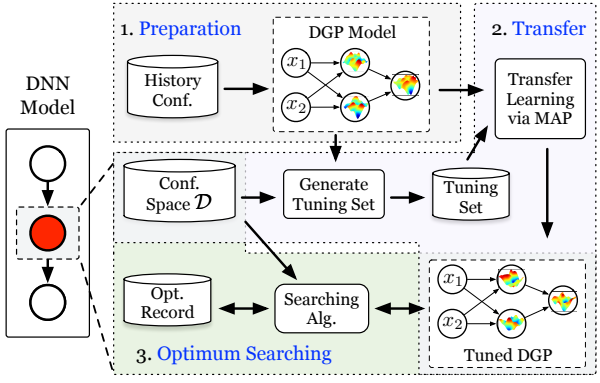
Transfer Knowledge – Maximum-a-posteriori Estimation

- Prior parameters $\tilde{\boldsymbol{\theta}}$, new parameters $\hat{\boldsymbol{\theta}}$
- According to the Bayes' theorem, find the optimal $\hat{\boldsymbol{\theta}}$, to maximize the posterior distribution $\mathcal{P}(\hat{\boldsymbol{\theta}}|\boldsymbol{y}^t)$:

$$\mathcal{P}(\hat{\boldsymbol{\theta}}|\boldsymbol{y}^t) \propto \mathcal{P}(\hat{\boldsymbol{\theta}}) \cdot \mathcal{P}(\boldsymbol{y}^t|\hat{\boldsymbol{\theta}}), \tag{2}$$

Our Solution – Stage 3: Optimal Searching

- The tuned DGP model is served as the performance estimator.
- No need to compile and deploy configurations.
- Use traditional searching algorithms (*e.g.*, simulated annealing) to search the optimal configuration.



Experimental Results

Experimental Settings

Baselines

- AutoTVM [1]
- CHAMELEON (ICLR'20) [2]
- GGA (DAC'20) [3]

Quality Metrics

- Inference latency
- Search time: time cost to find the best solution
- Hyper-volume (HV) = Redu. of Inference Latency $\times$ Redu. of Search Time $\times$ 100

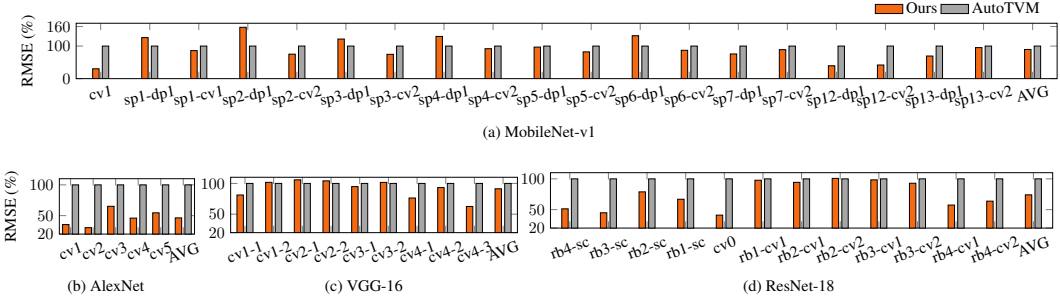
[1] Chen, Tianqi, et al. "Learning to optimize tensor programs." NeurIPS, 2018.

[2] Ahn, Byung Hoon, et al. "Chameleon: Adaptive code optimization for expedited deep neural network compilation." ICLR, 2020.

[3] Mu, Jiandong, et al. "A history-based auto-tuning framework for fast and high-performance DNN design on GPU." DAC, 2020.

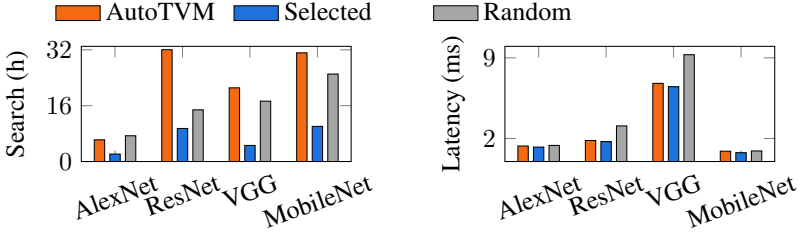
Ablation Studies on the Proposed DGP

- The pre-trained DGP model and XGBoost (used in AutoTVM) are directly used to predict the GFLOPS values of the new deployment tasks.
- RMSE (root-mean-square-error) is used to characterize the prediction errors.



Ablation Studies on the Transfer Learning

- AutoTVM
- Transfer learning based on the randomly-sampled tuning set
- Transfer learning based on the tuning set which is sampled by our pre-trained DGP



(a) Search Time

(b) Inference Latency

Experimental Results

Table: Comparisons of Search Time and End-to-end Model Inference Latency

Model	AutoTVM		CHAMELEON (ICLR'20)			Ours				
	Search (h)	Inference (ms)	Search Redu. (%)	Inference Redu. (%)	HV	Search (h)	Search Redu. (%)	Inference (ms)	Inference Redu. (%)	HV
MobileNet-v1	31.14	0.8980	-	-	-	10.06	67.69	0.7664	**14.65**	**9.9168**
AlexNet	6.28	1.3467	72.16	5.88	4.2409	2.14	65.96	1.2537	**6.91**	**4.5573**
VGG-16	19.92	6.7847	82.56	3.44	2.8418	4.61	76.83	6.4972	**4.24**	**3.2556**
ResNet-18	32.04	1.8248	76.67	4.16	3.1915	9.47	70.43	1.7305	**5.17**	**3.6423**

- DAC'20 tests on ResNet-18, reduces the search time by 93.17% and reduces the inference latency by 3.26%. HV is 3.0307, worse than ours and ICLR'20.
- Our method can achieve best final deployment performance and accelerate the search process at the same time.

THANK YOU!