Research on Deployment and Acceleration Optimization based on tvm

Yilin An

Abstract— **In recent years, with the continuous development and progress of machine learning, the field of deep learning has achieved a series of unprecedented successes, and deep learning algorithms have also been applied to all walks of life. But as the scale of the network becomes huge, there is a huge demand for computing power. Most deep learning frameworks, such as Tensorflow, MXNet, Pytorch, etc., only provide some server-level optimization. As a result, it is difficult to deploy deep neural network effectively on some devices with insufficient resources. Therefore, in the embedded field, relevant researches have been carried out one after another. As an important branch of deep learning, deep neural network has become more and more difficult to deploy because of its increasing computation. Based on the open source framework of TVM, this paper proposes a method to optimize the network computation diagram to accelerate the execution of neural network computation. This article uses YOLOV4-tTiny deployed on Jeston Nano, which is 13 times faster with ansor technology.**

Index Terms—**Machine Learning, TVM, YOLOV4-Tiny, Jeston Nano.**

I. Introduction

Deep learning is an algorithm that learns data representations at multiple levels of abstraction through a multi-layer network. It is a branch of machine learning and one of the most popular branches at the moment. Over the past decade, we have seen significant gains in deep learning in computer vision, speech recognition, natural language processing, and more. Deep learning has pushed many AI technologies to the point of application. Among them, face recognition [1] and recommendation system have done well in the implementation of the business. Medical care [2], agriculture, security and finance are also important application scenarios of deep learning. Although deep learning technology shows strong advantages and scalability in many fields, in recent years, the speed of theoretical research on deep learning tends to slow down, and reliability problems in some application fields such as automatic driving [3] have not been solved. Whether deep learning can maintain its current popularity and achieve rapid development in the future is closely related to whether more deep learning applications can be implemented. Different from traditional software and hardware algorithms that solve specific tasks, deep learning algorithms based on deep neural networks rely on learning knowledge from a large amount of training data. This kind of learning mode to match a certain pattern from a large amount of data has a very high demand on the computing power and storage of hardware. Moreover, the improvement of the accuracy of the deep learning model is often accompanied by the expansion of the network scale. With the expansion of the network scale, the corresponding demand for computing power and storage will be greater. The research shows that the computing required for training of the largest deep neural network model since 2012 has increased exponentially, doubling every 3.4 months.

However, with the popularization of deep learning applications, most of the current computing needs are model reasoning rather than model training. The computing needs of model reasoning will expand rapidly with the increase of business scenarios and the number of users. The reasoning environment of deep learning is more complex than that of training. At present, deep learning applications are widely deployed in various computing platforms including mobile devices, embedded devices, robot systems, drones, autonomous vehicles, and even single-chip computers. Most embedded computing platforms have limited computing power, storage, and power compared to PCS and servers, so many models are difficult to deploy directly on these platforms.

In order to cope with the storage and low-power computing problems of deep learning model on embedded devices, one approach is to modify the model so that it can adapt to resource-constrained computing platforms. Specifically, it includes designing lightweight models more suitable for mobile and embedded devices, with smaller model size and acceptable accuracy, which can be easily deployed on existing mobile devices. The model compression methods include quantization, pruning and knowledge distillation.To sum up, with the wide application of deep learning, the efficient and simplified deployment of neural networks has become a key issue. This paper takes the optimization deployment of deep learning model as the starting point, designs and implements compilers for deployment of different deep learning frameworks, and realizes the function of efficiently mapping each computing task in the neural network model to each execution unit of hardware.

II. RELATED WORK

A. *AI Compiler*

In order to better adapt to the deep learning framework and new deep learning hardware, academia and industry have proposed different deep learning compilers. At present, the mainstream deep learning compiler TVM can help developers compute more efficiently on different hardware. Tensor machine learning provides frame-independent

Manuscript received Dec 28, 2022
Yilin An, School of Computer Science and Technology, Tiangong University, Tianjin, China.

abstraction for high performance machine learning, Halide aims to make writing high performance graphics and array processing code much easier on modern machines, TVM [4]. and Tensor sions were both developed based on Halide, MLIR [5] is a new compiler infrastructure that dramatically reduces definition and entry costs, introduces a new level of abstraction for building dome-specific compilers, and is part of the LLVM [6] project.

Deep learning compilers typically include a front end and a middle end and a back end. The front end can be a new DSL (domain-specific language) that expresses optimization information available to the middle and back ends, such as Halide's separation of scheduling from algorithm, TVM's separation of schdule from compute, and Halide's separation of Scheduling from Algorithm. The Einstein expression in the Tensor Comprehension [7]; The middle end can be operator fusion, such as conv and add, conv and batchnorm fusion, removal of common subexpression, dead code elimination, and calculation graph or IR (intermediate representation) deformation, as long as the semantic is unchanged, other can change; The backend is related to the architecture. We can tiling or tensorizing the on-chip cache, using polyhedron or TVM ideas. For the optimization of parallel operation, there are various scheduling; For storage access optimization, there can be data arrangement changes, various hardware-specific optimizations, autotuning techniques, and optimized kernel libraries. Hardware-specific optimizations can efficiently generate code for different hardware goals.

In order to break down a problem into multiple steps to solve it, and break down a complex problem into multiple sub-problems, the current deep learning compilers all adopt multi-level IR processing. Taking TVM as an example, a high-level IR(Relay IR [8]) is designed, which is responsible for abstracting hardware-independent graph structure. In order to overcome the limitations of the expression of complex calculations used in the IR constrained deep learning model adopted by traditional compilers, existing deep learning compilers utilize advanced IR with a special design (called graphical IR) for efficient code optimization. Low-level IR(TIR) Hardware-related IR representation, implementation of low-level IR. The low-level IR describes the calculations of the deep learning model in a more granular representation than the high-level IR, which provides interfaces to fine-tune the calculations and memory access for goal-specific optimizations, and the compiler front end is responsible for hardware-independent optimizations based on the high-level IR. The compiler back end is responsible for hardware-specific optimization, code generation, and compilation based on low-level IR.

III. RELATED RESEARCH PROGRESS

A. ANSOR [9]

Ansor automatic tuning process, as shown in Figure 3-1, generates sketch extraction of high-level characteristics of the subgraph operator, performs coarse-grained optimization of the operator, determines the rough structure of the code, and randomly determines some split strategies and optimization strategies for the for cycle. Finally, we use the evolutionary search method to evaluate the performance using the cost function to get the optimal configuration for the hardware and then generate the corresponding code for the device. Each of these steps is described below:

Program sampling:

Ansor automatically expands the search space by recursively applying a flexible set of derivation rules and randomly sampling complete programs in the search space. Since random sampling provides an equal chance for each point to be sampled, the search algorithm can potentially explore every program in the space under consideration. You don't rely on random sampling to find the best program, because fine-tuning is done after each sampling program.

Sketch generation:

In sketch generation, the calculated graph is first sorted topologically and converted into a calculated queue, which is convenient for subsequent static analysis and scheduling strategy selection. Static analysis is used to extract operator features, which is a crucial step and also an innovation of Ansor. Ansor does not focus on processing a certain operator type, but extracts operator characteristics.

Random comment:

The resulting sketches are incomplete programs because they only have tile structures without specific tile dimensions and loop comments. The sketches need to be annotated so that they become complete programs for fine-tuning and evaluation. From the generated sketch list, randomly select a sketch, fill the size of the split, parallelize some outer loops, vectorize some inner loops, and expand some inner loops. In addition, the calculation position of some nodes in the program is changed randomly, and the split structure is fine-tuned. All "random" represents a uniform distribution of all valid values. If some particular algorithm requires custom annotations to take effect (for example, special expansion), allow the user to give a simple prompt in the calculation definition to adjust the annotation policy. Finally, because changing the layout of constant tensors can be done at compile time with no runtime overhead, rewrite the layout of constant tensors according to a multilevel split structure to make it as cache-friendly as possible. This optimization is very efficient because the weight tensor of the convolution or fully connected layer is a static tensor. Annotations are only responsible for randomly generating code, regardless of performance, which is guaranteed by performance fine-tuning.

Performance tuning:

The program sampled by the program sampler has good coverage of the search space, but the quality is not guaranteed. This is because the optimization selects random sampling. The performance debugger fine-tunes the sampler's performance through evolutionary search and a learnable cost model.

Fine-tuning is performed iteratively. In each iteration, evolutionary search is first used to find a small set of better-performing programs based on the learned cost model. The actual execution time of these programs is then measured on the hardware. Finally, the analytical data obtained from the measurements were used to retrain the cost model to make it more accurate.

The calculation to be optimized may be very complex, and it is not practical to walk through all the comments to obtain the optimal implementation. Ansor trains an

XGBoost tree (XGBoost is an optimized distributed gradient enhancement library) to evaluate the code performance, compared with the results obtained by running directly at runtime. The trained XGBoost [10] can determine roughly how good the code is, and it takes less time. To evaluate the performance of code generated by annotations, evolutionary search uses a random sample of programs as well as high-quality programs from the previous evaluation as the initial population, and applies variation and crossover to generate the next generation. The learnable cost model is used to predict the performance of each program, such as throughput. Run the evolution of fixed algebra and select the best program in the search process. Because the cost model can give relatively accurate estimates of program performance while being several orders of magnitude faster than actual measurements, it enables us to compare tens of thousands of programs in the search space in a matter of seconds and pick out good performance programs for evaluation.

IV. EXPERIMENT

A. *Experimental Process*

In the first step, we conducted training for YOLOv4-Tiny and YOLOX-s network with image size of 416 on the server, using masic data enhancement, label smoothing, and training skills of learning rate cosine annealing attenuation. Network after 100 rounds of training, confidence set to 0.5, nms_iou set to 0.3, Batch_size equal to 32, learning rate 0.001, loading pre-training weight, after 50 rounds of freezing training and 50 rounds of thawing training, the second part for training good floating point weight after static quantization, We fed 5000 images to determine the scaling factor and bias of each layer. In the third step, the weight was loaded into TVM for further graph optimization, and the optimized calculated graph was used for inference.

B. *Dataset*

Dataset adopted PASCAL VOC2007[62] data set, VOC data set included: 5011 training sets, 4952 test sets, a total of 9963 sets, including 20 categories, the category mainly includes 8 kinds of Vehicle, 5 kinds of Household, 6 kinds of Animal, Person as a separate category, a total of 20 categories. The class information of PSSCAL VOC2007[64] is shown in Table 5-1. The ratio of (training set + verification set) to test set in PASCAL VO2007C is 9:1, and the ratio of training set to verification set in (training set + verification set) is 9:1. Dataset category shows in TAB.1.

TAB.1 Datasets category

Vehicles	Household	Animals	Person
car	chair	cat	--
bus	sofa	dog	--
bicycle	tv/monitor	cow	--
motorbike	bottle	horse	--
boat	Potted plant	sheep	--
train	--	bird	--
aeroplane	--	--	--
boat	--	--	--

C. *Experimental Result*

Experimental results show that the yolov4-tiny network is deployed on jetson nano after ansor selection with optimization level equal to 3 acceleration. After the Trial of 200 tuning, the speed increased by 5.55 times . After the Trial of 2000 tuning, the speed increased by 9.96 times . After the Trial of 10000 tuning, the speed increased by 13 times . The acceleration effect is shown in TAB.2.

TAB.2 acceleration effect

	FPS	mAP(%)	Speed up
Baseline	13.5	83.8%	
Trial=200	74.925	83.6%	5.55
Trial=2000	134.5	83.5%	9.96
Trial=10000	175.5	83.3%	13

V. CONCLUSION

In this paper, to solve the problem of slow reasoning speed of the model, TVM is used for graph optimization, ansor's optimization search strategy is adopted for space exploration, and the configuration found is used for deployment, which is deployed on Jeston Nano. After the optimization of automatic tuning technology, the speed can be increased by 13 times after 10,000 trials.

REFERENCES

[1] Deng J, Guo J, An X, et al. Masked face recognition challenge: The insightface track report[C]//Proceedings of the IEEE/CVF International Conference on Computer Vision. 2021: 1437-1444.
[2] Shaheen M Y. Applications of Artificial Intelligence (AI) in healthcare: A review[J]. ScienceOpen Preprints, 2021.
[3] Zhang C, Lu Y. Study on artificial intelligence: The state of the art and future prospects[J]. Journal of Industrial Information Integration, 2021, 23: 100224.
[4] Chen T, Moreau T, Jiang Z, et al. {TVM}: An automated {End-to-End} optimizing compiler for deep learning[C]//13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18). 2018: 578-594.
[5] Lattner C, Amini M, Bondhugula U, et al. MLIR: A compiler infrastructure for the end of Moore's law[J]. arXiv preprint arXiv:2002.11054, 2020.
[6] Zakowski Y, Beck C, Yoon I, et al. Modular, compositional, and executable formal semantics for LLVM IR[J]. Proceedings of the ACM on Programming Languages, 2021, 5(ICFP): 1-30.
[7] Vasilache N, Zinenko O, Theodoridis T, et al. Tensor comprehensions: Framework-agnostic high-performance machine learning abstractions[J]. arXiv preprint arXiv:1802.04730, 2018.
[8] Roesch J, Lyubomirsky S, Weber L, et al. Relay: A new ir for machine learning frameworks[C]//Proceedings of the 2nd ACM

SIGPLAN international workshop on machine learning and programming languages. 2018: 58-68.

[9] Zheng L, Jia C, Sun M, et al. Ansor: Generating {High-Performance} Tensor Programs for Deep Learning[C]//14th USENIX symposium on operating systems design and implementation (OSDI 20). 2020: 863-879.

[10] Chen T, Guestrin C. Xgboost: A scalable tree boosting system[C]//Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining. 2016: 785-794.