جست‌وجوی معماری شبکه‌های عصبی عمیق آگاه از منابع در سامانه‌های نهفتهٔ چندهسته‌ای

سهیل رستاری[۱]، مرتضی محجل کفشدوز*[۲]، محبوبه شمسی[۳]
کارشناس ارشد دانشکدهٔ برق و کامپیوتر، دانشگاه صنعتی قم، قم، ایران[۱]
استادیار، دانشکدهٔ برق و کامپیوتر، دانشگاه صنعتی قم، قم، ایران*[۲]
دانشیار، دانشکدهٔ برق و کامپیوتر، دانشگاه صنعتی قم، قم، ایران[۳]

چکیده

ایجاد شبکه‌های عصبی به‌صورت دستی فرایندی کند و مبتنی بر آزمون‌وخطا است که با افزایش تعداد پارامترها یا لایه‌ها، به‌طور قابل‌توجهی هزینه‌بر و غیربهینه می‌شود. برای حل این مشکل، الگوریتم‌های خودکار جست‌وجوی معماری شبکه (NAS) معرفی شده‌اند که به‌تازگی توانسته‌اند در مجموعه‌داده‌های مختلفی مانند CIFAR-10، ImageNet و Penn Tree Bank به دقت‌های بالایی دست یابند. این الگوریتم‌ها قادرند فضای وسیعی از معماری‌ها با ویژگی‌های متنوع مانند عمق، عرض، نوع اتصالات و عملیات‌ها را جست‌وجو و معماری‌های بهینه را کشف کنند؛ بااین‌حال، یکی از چالش‌های اصلی NAS زمان جست‌وجوی طولانی آن‌هاست که ممکن است به ده‌ها هزار ساعت GPU برسد؛ هرچند با پژوهش‌های اخیر این زمان به ده‌ها ساعت کاهش یافته‌است؛ علاوه‌بر این، این روش‌ها بر حسب معمول تنها بر بهبود دقت شبکه تمرکز می‌کنند و به معیارهای مهمی مانند سرعت شبکه و منابع مصرفی توجهی ندارند. این مسئله استفادهٔ مستقیم از آن‌ها را در سامانه‌های نهفته با محدودیت منابع مانند قدرت پردازشی، حافظه و انرژی مصرفی دشوار می‌سازد؛ بنابراین، نیاز به روش‌های جست‌وجوی آگاه از محدودیت‌های سامانه وجود دارد. در این مقاله، روشی بر اساس کاهش گرادیان ارائه می‌شود که به‌طور خودکار شبکه‌هایی مناسب برای اجرا بر روی پردازنده‌های چندهسته‌ای بدون GPU طراحی می‌کند. در این روش، یک اَبَرشبکه با مسیرهای موازی و بلوک‌های محاسباتی ایجاد می‌شود. برای انتخاب عملیات مناسب در هر بلوک از مسیر، از تعدادی متغیر تصمیم‌گیری استفاده می‌شود؛ همچنین، برای بهره‌مندی مسیرهای موازی از نتایج میانی یکدیگر و افزایش دقت شبکه، علاوه‌بر تصمیم‌گیری در مورد عملیات هر بلوک، دربارهٔ نقاط همگام‌سازی نیز تصمیم‌گیری می‌شود؛ سپس با آموزش متغیرهای تصمیم‌گیری (نوع بلوک و نقاط همگام‌سازی) هم‌زمان با وزن‌های اصلی شبکه، زیرشبکه مناسبی انتخاب می‌شود. در روش ارائه‌شده به‌دلیل استفاده از روش کاهش گرادیان، تنها دو بار فرایند آموزش انجام می‌شود. این امر زمان اجرا را نسبت به روش‌های مبتنی بر جست‌وجوی تکاملی و یادگیری تقویتی کاهش قابل‌توجهی می‌دهد؛ علاوه‌بر این، با درنظرگرفتن محدودیت‌های سامانهٔ هدف مانند تعداد هسته‌ها و حافظهٔ مصرفی، می‌توان به معماری مناسب‌تری نسبت به سایر روش‌ها دست یافت. آزمایش‌های انجام‌شده بر روی مجموعه‌دادهٔ CIFAR-10 نشان می‌دهد که روش پیشنهادی می‌تواند با زمان جست‌وجوی بسیار کم به دقت مناسبی دست یابد که کارایی آن را برای سامانه‌های نهفته با منابع محدود تأیید می‌کند.

واژگان کلیدی: جست‌وجوی معماری شبکه‌های عصبی، سامانه‌های نهفته، موازی‌سازی، پردازنده‌های چندهسته‌ای، روش کاهش گرادیان.

Resource-Aware Neural Architecture Search for Multicore Embedded Real-Time Systems

Soheil Rastari[1], Morteza Mohajjel Kafshdooz[*2], Mahboubeh Shamsi[3]
Master, Faculty of Electrical and Computer Engineering, Qom University of Technology, Qom, Iran[1]
Assistant Professor, Faculty of Electrical and Computer Engineering, Qom University of Technology, Qom, Iran[*2]
Associate Professor, Faculty of Electrical and Computer Engineering, Qom University of Technology, Qom, Iran[3]

Abstract

Creating neural networks in a non-automatic way is a slow process based on trial and error. When the number of network parameters or the number of layers increases, the non-automatic method becomes very expensive and the final result may be suboptimal. Automatic network architecture search algorithms are used to solve this problem. Recently, these algorithms have been able to achieve high accuracies on various datasets such as

* نویسندهٔ عهده‌دار مکاتبات
* Corresponding author

• تاریخ ارسال مقاله: ۱۴۰۲/۱۱/۳ • تاریخ پذیرش: ۱۴۰۳/۱۲/۱۸ • تاریخ انتشار: ۱۴۰۴/۳/۲۸ • نوع مطالعه: پژوهشی

[DOI: 10.61186/jsdp.22.1.25]

CIFAR-10, ImageNet, and Penn Tree Bank. These algorithms have the ability to search a wide space of architectures with different characteristics such as network depth, width, connection method, and operations in order to discover architectures with appropriate accuracy. However, one of the traditional challenges of these algorithms is their high search time (Approximately tens of thousands of GPU hours), which has been reduced to tens of hours with new research. Another challenge that usually exists in these methods is their focus on improving network accuracy, while other criteria such as network speed and consumed resources are not taken into account. As a result, these methods cannot be used directly to find the optimal architecture in embedded systems that have limited resources such as processing power, memory, and energy consumption. Therefore, search methods should be devised that are aware of these limitations. Research has been done in this field in recent years, but these methods do not focus specifically on coarse-grained multi-core architectures that do not have a GPU. In this article, we present a method for the automatic design of networks that are suitable for running on multi-core processors. In this method, based on gradient descent, a SuperNet with parallel paths and computational blocks is created. The number of parallel paths is equal to or less than the number of cores. We use a series of decision variables to select appropriate operations in each block of the path. In addition to deciding on the operations performed in each block, deciding is also made regarding synchronization points to utilize the intermediate results of parallel paths and improve the network's accuracy. Then, by training the decision variables (block type and synchronization points) simultaneously with the main network weights, an appropriate subnetwork is selected. Due to the use of the gradient descent method in this approach, the training process is performed only twice, resulting in the final structure of the network. As a result, it has a much lower execution time compared to other methods based on evolutionary search and reinforcement learning. Additionally, considering the constraints of the target system, such as the number of cores and memory consumption, can lead to a more suitable architecture compared to other methods. Experiments conducted on the CIFAR-10 dataset demonstrate that the proposed method can achieve satisfactory accuracy with very little search time.

Keywords: Neural network architecture search, embedded systems, parallelization, multi-core processors, gradient descent method.

۱-مقدمه

یادگیری عمیق[1] در طول سال‌های گذشته پیشرفت قابل توجهی در زمینه‌های مختلف مانند تشخیص تصویر[2]، تشخیص گفتار[3] و ترجمه ماشینی[4] داشته‌است[۱]؛ یکی از دلایل مهم این پیشرفت‌ها، معماری‌های عصبی جدید است. معماری‌های مورد استفاده در حال حاضر بیشتر به‌صورت غیرخودکار توسط متخصصان انسانی توسعه داده شده‌اند که فرایندی زمان‌بر و مستعد خطا است[۲]. در چند سال گذشته، تلاش‌های پژوهشی زیادی برای خودکارسازی این فرایند انجام شده‌است.

جست‌وجوی معماری عصبی[5] **(NAS)** روش یا فرایندی است که برای طراحی یا کشف خودکار معماری شبکه‌های عصبی بهینه استفاده می‌شود. روش‌های جست‌وجوی معماری عصبی در چند سال گذشته به پیشرفت‌های چشم‌گیری در زمینهٔ بهبود دقت، کاهش هزینه‌های محاسباتی و حافظهٔ مصرفی رسیده‌است و درنهایت توانسته‌است از عملکرد انسان در طراحی معماری عصبی پیشی بگیرد[۳].

همان‌طورکه در شکل (۱) نشان داده شده‌است، فرایند جست‌وجوی معماری عصبی به سه بخش تقسیم می‌شود:

فضای جست‌وجو[6]: به فضای حالت ممکن از انواع معماری‌های شبکه که در هنگام اجرای الگوریتم جست‌وجو قابل انتخاب است، گفته می‌شود. انتخاب صحیح فضای جست‌وجو تأثیر بسیار قابل توجهی در کاهش زمان جست‌وجو و همچنین رسیدن به اهداف جست‌وجو (مانند دقت شبکه، کاهش حافظهٔ مصرفی و محاسبات) دارد. فضای جست‌وجو به‌طور معمول مجموعه‌ای از عملیات متعارف موجود در شبکه‌های عصبی مانند کانولوشن[7]، کامل متصل[8]، ادغام[9] و ترتیب و نحوهٔ اتصال این عملیات است[۴].

الگوریتم جست‌وجو[10]: این مؤلفه نحوهٔ کاوش فضای جست‌وجو برای یافتن معماری مناسب را تعیین می‌کند[۳]. الگوریتم جست‌وجو بر اساس معیارهای مشخص‌شده به جست‌وجوی فضای حالت می‌پردازد. جست‌وجو یا به صورت گسسته[11] مانند الگوریتم‌های تکاملی[12][۵] و یادگیری تقویتی[13][۶] و یا پیوسته[14] مانند الگوریتم‌های کاهش گرادیان[۷] انجام می‌شود.

[1] Deep learning
[2] Image recognition
[3] Speech recognition
[4] Machine translation
[5] Neural architecture search

[6] Search space
[7] Convolution
[8] Fully-connected
[9] Pooling
[10] Search algorithm
[11] Discrete
[12] Evolutionary algorithm
[13] Reinforcement learning
[14] Continuous

[DOI: 10.61186/jsdp.22.1.25]

(شکل-۱): فرایند جست‌وجوی معماری عصبی.

(Figure-1): Neural architecture search process.

در الگوریتم‌های تکاملی که از طبیعت الهام گرفته شده‌اند، یک فضای جست‌وجوی شبکه با عملیات و اتصالات مختلف ایجاد می‌شود؛ سپس با عمل‌گرهای جهش[۱] و تقاطع[۲] در چندین دور نامزدهای معماری بهتری را تولید می‌کند و درنهایت یکی از معماری‌هایی را که عملکرد بهتری دارد، به‌عنوان معماری هدف انتخاب می‌کند[۲].

در روش‌های مبتنی بر یادگیری تقویتی، عامل در محیط، فضای معماری‌های ممکن شبکه را جست‌وجو کرده و از آن‌ها نمونه‌برداری می‌کند؛ همچنین معیار عملکرد معماری‌های نامزد را به‌عنوان پاداش برای ارزیابی نگهداری می‌کند و سرانجام بر اساس خط‌مشی (بر اساس اقدامات و پاداش‌های گذشتهٔ خود) که در شبکه دارد، معماری‌های نامزد که بالاترین پاداش را دارند به‌عنوان معماری نهایی انتخاب می‌کند[۳].

در روش‌های مبتنی بر کاهش گرادیان[۳] فضای جست‌وجو به‌صورت یکسری متغیر تصمیم‌گیری مدل می‌شود و مقدار این متغیرها هم‌زمان با سایر متغیرهای شبکه در حین آموزش شبکه با الگوریتم‌های رایج آموزش شبکه‌های عصبی مانند کاهش گرادیان تصادفی[۴]، آدام[۵] و غیره به‌دست می‌آیند[۱].

روش ارزیابی[۶]: در این بخش عملکرد شبکه‌های عصبی نامزد ارزیابی می‌شود[۳]. برای ارزیابی دقیق یک شبکهٔ عصبی فرایند آموزش باید به صورت کامل انجام شود و شبکه بر روی سخت‌افزار نهایی ارزیابی شود، اما فرایند آموزش بسیار زمان‌گیر است و اجرا بر روی سخت‌افزار نهایی علاوه‌بر زمان‌بربودن چالش‌های مخصوص به خود را دارد؛ از طرف دیگر تعداد اجرای عملیات ارزیابی به‌ویژه در روش‌های تکاملی و روش‌های یادگیری تقویتی زیاد است؛ درنتیجه بر حسب معمول از روش‌های تخمینی به‌جای مقادیر دقیق معیارها استفاده می‌شود؛ البته با توجه به این‌که در روش کاهش گرادیان به‌طور معمول تنها دو بار فرایند آموزش انجام می‌شود، می‌توان در این روش از تخمین‌های دقیق‌تر استفاده کرد[۲].

جست‌وجوی معماری عصبی در سامانه‌های نهفته[۷] به فرایند طراحی خودکار معماری‌های شبکهٔ عصبی اشاره دارد که برای استقرار در سامانه‌های نهفته با محدودیت منابع مناسب‌اند. سامانه‌های نهفته، مانند دستگاه‌های اینترنت اشیا[۸]، تلفن‌های همراه یا دستگاه‌های محاسباتی لبه[۹]، منابع محاسباتی محدودی از جمله حافظه، قدرت پردازش و انرژی مصرفی دارند[۸].

تمرکز اصلی جست‌وجوی معماری عصبی در سامانه‌های نهفته، طراحی معماری شبکه‌های عصبی است که از نظر محاسباتی کارآمدند، حافظهٔ کم و کمترین انرژی را مصرف می‌کنند[۳]؛ برای این منظور باید معماری‌هایی را انتخاب کرد که بتواند تعادل مناسبی بین دقت و کارایی ایجاد کند. بسیاری از سامانه‌های نهفته نیاز به استنتاج[۱۰] با تأخیر کم یا پردازش بی‌درنگ[۱۱] دارند. جست‌وجوی معماری عصبی این محدودیت‌ها را درنظر می‌گیرد و معماری‌هایی را طراحی می‌کند که می‌توانند زمان استنتاج یا زمان پاسخ مورد نیاز را برآورده کنند[۹].

در سال‌های اخیر، چندین معماری شبکهٔ عصبی کانولوشنی کارآمد و فشرده برای دستگاه‌های لبه طراحی شده‌است؛ از جمله MobileNets [۱۰]، ShuffleNets [۱۱] و EfficientNet [۱۲]؛ درحالی‌که این مدل‌ها بسیار کارآمدند، اما نیاز به تنظیم دستی طولانی و زمان‌بر فراپارامترها[۱۲] دارند. برای حل این مسئله، بسیاری از روش‌های خودکار یا نیمه‌خودکار برای بهینه‌سازی معماری شبکه‌های عصبی، که زمان طراحی را کاهش می‌دهد،

[1] Mutation
[2] Crossover
[3] Gradient decent
[4] Stochastic Gradient Descent
[5] ADAM
[6] Evaluation method

[7] Embedded systems
[8] Internet of Things
[9] Edge
[10] inference
[11] Real-time processing
[12] Hyper Parameters

[DOI: 10.61186/jsdp.22.1.25]

پیشنهاد شده‌اند؛ همان‌طورکه ذکر شد الگوریتم‌های جست‌وجوی شبکه‌های عصبی به سه دستهٔ کلی: ۱) روش‌های تکاملی، ۲) روش یادگیری تقویتی و ۳) روش‌های کاهش گرادیان تقسیم‌بندی می‌شوند.

الگوریتم یادگیری تقویتی و الگوریتم تکاملی به‌دلیل گسسته‌بودن فضای جست‌وجو، تعداد زیادی از معماری‌های نامزد را ارزیابی می‌کنند. ارزیابی هر یک از نامزدها بر حسب معمول نیاز به آموزش شبکه دارد که بسیار زمان‌بر است؛ درنتیجه به‌طور معمول زمان اجرای این الگوریتم‌ها به‌شدت زیاد بوده و به منابع محاسباتی زیادی نیاز دارد. در سال‌های اخیر برای حل مسئلهٔ زمان جست‌وجوی روش‌های یادگیری تقویتی و تکاملی، از روش کاهش گرادیان و یا به عبارت دیگر جست‌وجوی شبکهٔ عصبی مشتق‌پذیر[1] استفاده می‌شود.

جست‌وجوی شبکهٔ عصبی مشتق‌پذیر با استفاده از روش بهینه‌سازی گرادیان فضای جست‌وجو را پیوسته می‌کند و درنتیجه زمان جست‌وجو کاهش می‌یابد. نخستین مقاله‌ای که از این روش برای جست‌وجوی شبکهٔ عصبی استفاده کرد مقاله DARTS[2] [۷] بود. پس از ارائه این مقاله مقالات مشابه دیگری نیز ارائه شدند که سعی در بهبود عملکرد آن داشتند[۱۳، ۱۴، ۱۵، ۱۶، ۱۷]؛ برای مثال روش ProxylessNAS [۱۳] از نخستین روش‌هایی بود که پس از روش DARTS ارائه شد. این روش برای حل مشکل حافظهٔ مصرفی بالا، از روش‌های مبتنی بر کاهش گرادیان استفاده می‌کرد تا در هر زمان تنها یکی از زیرشبکه‌ها در حافظهٔ آموزش داده شود؛ برای این منظور در این روش متغیرهای تصمیم‌گیری معماری شبکه دودویی[3] می‌شدند و درنتیجه تنها یک مسیر در هر زمان فعال بود.

روش FBnet [۱۴] یکی دیگر از این روش‌هاست که بر طراحی شبکه‌های عصبی کانولوشنی کارآمد که از محدودیت‌های سخت‌افزاری آگاه‌اند تمرکز دارد. در این روش بر خلاف روش‌های پیشین به تأخیر[4] شبکه علاوه‌بر حجم محاسبات نیز توجه شده‌است. برای محاسبهٔ تأخیر شبکه، تأخیر هر یک از عملیات در سخت‌افزار هدف مانند تلفن‌های همراه اندازه‌گیری می‌شود؛ سپس این مقادیر با هم جمع شده و تأخیر کل شبکه به‌دست می‌آید. در روش FBNetV2 [۱۵] فضای جست‌وجوی روش FBnet گسترش داده شد.

یکی از مسائلی که در این روش‌ها (دست‌کم به‌صورت مستقیم) به آن پرداخته نشده‌است درنظرگرفتن معماری‌های شبکه‌ای است که مناسب اجرا بر روی سخت‌افزارهای چندهسته‌ای باشند. پردازندهٔ چندهسته‌ای به‌دلیل موازات درشت‌دانه (برخلاف پردازنده‌های گرافیکی که موازات ریزدانه دارند) قابلیت مناسبی در اجرای شبکه‌های موازی دارند؛ بنابراین درصورتی‌که سخت‌افزار هدف پردازنده‌های چندهسته‌ای باشند به نظر می‌رسد در فضای جست‌وجو باید شبکه‌هایی با معماری موازی نیز درنظر گرفته شود.

در این مقاله به ارائهٔ روشی جهت طراحی خودکار شبکه‌های موازی که مناسب اجرا بر روی پردازنده‌های چندهسته‌ای هستند، خواهیم پرداخت. در این روش، یک آبَرشبکهٔ موازی از بلوک‌های محاسباتی ایجاد می‌کنیم که دارای لایه‌های موازی قابل انتخاب است. برای اینکه در حین فرایند آموزش بتوان لایه‌های مناسب را انتخاب کرد از متغیرهای تصمیم‌گیری که مقدار آن‌ها در حین آموزش شبکه مشخص می‌شود، استفاده می‌کنیم.

علاوه‌براین، برای اینکه شاخه‌های موازی شبکه بتوانند از ویژگی‌های استخراج‌شدهٔ همدیگر استفاده کنند در برخی از نقاط شبکه از عملیات همگام‌سازی استفاده می‌کنیم. با توجه به این‌که عملیات همگام‌سازی سربار زمانی دارد، برای نقاط همگام‌سازی نیز متغیرهای تصمیم‌گیری درنظر گرفته‌ایم؛ درواقع در زمان آموزش شبکه هم‌زمان با آموزش وزن‌های اصلی مشخص می‌شود: ۱) کدام لایه داخل بلوک‌های آبَرشبکه باشد و همچنین؛ ۲) در کدام نقطه از آبَرشبکه همگام‌سازی انجام شود. بعد از آموزش شبکه و انتخاب لایه‌ها و نقاط همگام‌سازی در آبَرشبکه، لایه‌ها و نقاط همگام‌سازی را که انتخاب‌نشدن هرس می‌کنیم و دوباره شبکهٔ نهایی را از مقادیر نهایی به‌دست‌آمده در مرحلهٔ پیش، آموزش می‌دهیم و پس از اجرای این مرحله شبکهٔ نهایی قابل استفاده بر روی سامانه‌های نهفته به‌دست می‌آید.

در این روش به‌دلیل استفاده از روش کاهش گرادیان تنها دو بار فرایند آموزش انجام می‌شود و ساختار نهایی شبکه به‌دست می‌آید و درنتیجه نسبت به سایر روش‌های مبتنی بر جست‌وجوی تکاملی و یادگیری تقویتی، زمان جست‌وجو بسیار کمتر خواهد شد. نکتهٔ قابل توجه در روش ارائه‌شده این است که در تابع ضرر مقدار هزینهٔ محاسباتی شبکه نیز درنظر گرفته می‌شود. آزمایش‌های انجام‌شده بر روی مجموعه‌دادهٔ CIFAR-10 نشان می‌دهد که روش ارائه‌شده می‌تواند به دقت مناسبی با زمان جست‌وجوی بسیار کم و همچنین هزینهٔ محاسباتی کمتری دست یابد. نتایج آزمایش‌ها در روش ارائه‌شده به دقت ۸۴ درصد می‌رسد که نسبت به سایر حالت‌ها حدود سه درصد بهبود یافته‌است. نوآوری اصلی مقاله

[1] Differentiable Neural Architecture Search
[2] Differentiable Architecture Search
[3] Binary
[4] Delay

[DOI: 10.61186/jsdp.22.1.25]

جست‌وجوی معماری بهینهٔ موازی با امکان انتخاب نقاط همگام‌سازی برای اجرا بر روی پردازندهٔ چندهسته با استفاده از روش گرادیان کاهشی است.

به‌اختصار کاری‌هایی که در این مقاله انجام داده‌ایم به این صورت است:

- از متغیرهای تصمیم‌گیری برای انتخاب لایه‌های کارآمد در داخل بلوک‌ها و لایه‌هایی که تعداد محاسبات ممیز شناور[1] (FLOPS) کمتری دارند استفاده می‌کنیم؛ همچنین:
- از متغیرهای تصمیم‌گیری برای انتخاب نقاط همگام‌سازی استفاده می‌کنیم که شبکه در زمان آموزش تشخیص می‌دهد نیاز به همگام‌سازی بین بلوک‌ها است یا نه.
- معماری‌های موازی را جست‌وجو می‌کنیم.
- در این روش به‌دلیل استفاده از روش کاهش گرادیان تنها دو بار فرایند آموزش انجام می‌شود و ساختار نهایی شبکه به‌دست می‌آید.

در ادامهٔ این مقاله، در بخش دوم به مرور کارهای گذشتهٔ جست‌وجوی معماری مشتق‌پذیر آگاه از سخت‌افزار خواهیم پرداخت. در بخش سوم به پیش‌نیازها و محدودیت‌های سامانه‌های نهفته می‌پردازیم. در بخش چهارم روش پیشنهادی را شرح می‌دهیم. در بخش پنجم نتایج آزمایش‌های مختلف و به همراه تحلیل آن‌ها توضیح داده می‌شود و درنهایت در بخش ششم به نتیجه‌گیری مطالب بیان‌شده می‌پردازیم.

۲- مرور پژوهش‌های پیشین

روش‌های جست‌وجوی خودکار شبکه‌های عصبی در کاربردهای مختلف مانند طبقه‌بندی تصویر به عملکرد مناسب رسیده‌اند. بهترین الگوریتم‌های جست‌وجوی معماری موجود برخلاف عملکرد قابل توجهشان، از نظر محاسباتی پرهزینه‌اند؛ برای مثال، دست‌یابی به یک معماری پیشرفته برای مجموعه‌داده‌های CIFAR-10 و ImageNet به دوهزار روز GPU برای الگوریتم یادگیری تقویتی یا ۳۱۵۰ روز GPU برای الگوریتم تکاملی نیاز دارد.

برای حل مشکلات روش‌های بالا همان طورکه در بخش مقدمه توضیح داده شد، روش کاهش گرادیان از نظر دقت و کاهش زمان آموزش بهتر است. در ادامه به بررسی روش‌های مبتنی بر کاهش گرادیان می‌پردازیم.

DARTS: فضای جست‌وجوی DARTS از تعدادی سلول محاسباتی[2] تشکیل شده‌است که این سلول‌ها را در زمان آموزش جست‌وجو می‌کند[۷]. یک سلول محاسباتی یک DAG[3] است که از تعدادی گره تشکیل شده‌است. هر گره نشان‌دهندهٔ یک نقشهٔ ویژگی[4] است. بین دو تا جفت گره تعدادی یال قرار گرفته‌است که نشان‌دهندهٔ عملیات‌هاست. هدف یادگیری الگوی اتصال و عملیات در هر گره است که عملکرد شبکه را به بیشینه می‌رساند[۱۸].

برای بهینه‌سازی جست‌وجوی معماری DARTS، از رویکرد بهینه‌سازی دوسطحی (سطح بالا و سطح پایین) استفاده شده‌است. در بهینه‌سازی سطح بالا، وزن‌های شبکه در زمان پس‌انتشار[5] بهینه می‌شوند. در بهینه‌سازی سطح پایین، معماری بهینه با کمینه‌کردن خطای مجموعهٔ اعتبارسنجی[6] از طریق کاهش گرادیان و تغییر متغیرهای معماری به‌دست می‌آید. در هر سلول معیار انتخاب یک عملیات بیشتربودن مقدار متغیر تصمیم‌گیری آن نسبت به سایر متغیرهای تصمیم‌گیری است[۷].

ProxylessNAS: از روش‌هایی مانند الگوریتم یادگیری تقویتی، الگوریتم تکاملی، یا الگوریتم کاهش گرادیان برای کاوش کارآمد فضای جست‌وجوی معماری استفاده می‌کند؛ در کل یک شبکهٔ بزرگ بدون هیچ فرا کنترل[7] با پارامترهای زیاد را آموزش می‌دهد که شامل تمام مسیرهای نامزد است؛ بااین‌حال، بودن مسیرهای نامزد منجر به سرریز[8] حافظهٔ GPU می‌شود[۱۳]؛ زیرا مصرف حافظه به طور خطی با تعداد مسیرهای نامزد افزایش می‌یابد؛ بنابراین، با هدف کاهش مصرف حافظهٔ GPU، متغیرهای تصمیم‌گیری معماری را دودویی می‌کند تا متوجه شود کدام مسیرها زائد است و تنها یک مسیر در هر زمان فعال باشد؛ با این کار مصرف حافظهٔ مورد نیاز در زمان آموزش بهv, قابل توجهی کاهش می‌یابد؛ اگر چهProxylessNAS زمان جست‌وجو را کاهش می‌دهد، اما همچنان به فضای جست‌وجوی بزرگی نیاز دارد. برای کاهش این پیچیدگی، Single-Path NAS [۱۹] پیشنهاد شده‌است.

Single-Path NAS: فضای جست‌وجوی Single-Path NAS از تعدادی عملیات کانولوشنی نامزد تشکیل شده‌است، تمام این عملیات‌های نامزد در زیرمجموعه‌های یک آبرهسته[9] ترکیب می‌شود. به جای درنظرگرفتن هر عملیات نامزد به‌طور جداگانه، در این روش از مجموعه‌ای مشترک از وزن‌ها به نام وزن هستهٔ کانولوشنی برای نمایش احتمالات عملیات‌های نامزد استفاده می‌کند. این بدان معناست که تمام عملیات‌های نامزد در یک آبرهستهٔ واحد رمزگذاری می‌شود و یک مسیر واحد را

[1] Floating-point operations per second
[2] Computing cell
[3] Directed acyclic graph
[4] Feature map
[5] Backpropagation
[6] Validation set
[7] Meta-Control
[8] Overflow
[9] Superkernel

[DOI: 10.61186/jsdp.22.1.25]

تشکیل می‌دهد. با انجام این کار، فرایند طراحی را ساده می‌کند؛ زیرا نیازی نیست که هر عملیات نامزد را جداگانه درنظر بگیرد و ارزیابی کند؛ ازاین‌رو تعداد پارامترهای قابل آموزش و هزینهٔ جست‌وجو به چند دوره کاهش می‌یابد[۱۹].

FBnet: این روش با یک فضای جست‌وجوی کوچک و کارآمد محاسباتی شروع می‌شود و به‌تدریج آن را گسترش می‌دهد تا بلوک‌های پیچیده‌تر و الگوهای اتصال بیشتری را دربربگیرد. این جست‌وجوی تدریجی امکان کاوش کارآمدتر در فضای معماری را فراهم می‌کند. در تابع ضرر FBNet علاوه‌بر دقت شبکه که به‌وسیلهٔ آنتروپی متقابل[1] درنظر گرفته می‌شود، تأخیر شبکه نیز درنظر گرفته می‌شود؛ به این صورت که اگر تأخیر شبکهٔ انتخاب‌شده بالا باشد، اثر خود را در بالارفتن مقدار ضرر می‌گذارد. برای تخمین تأخیر کل شبکه، تأخیر هر یک از عملیات به‌دست‌آمده و از جمع این تأخیرها، تأخیر کل شبکه به‌دست می‌آید[۱۴].

FBNetV2: این الگوریتم فضای جست‌وجو را نسبت به روش‌های پیشین گسترش می‌دهد و جست‌وجوی عرض و طول و همچنین تعداد کانال‌های نقشهٔ ویژگی را نیز در شبکه‌های عصبی کانولوشنی پشتیبانی می‌کند. این مقاله دو روش را پیشنهاد می‌کند: ۱) سازوکار پوشاندن[2] و ۲) انتشار شکل مؤثر[3]. سازوکار پوشاندن استفادهٔ دوبارهٔ نقشه‌های ویژگی بر اساس اهمیت یا ارتباط آن‌ها برای محاسبات بعدی را امکان‌پذیر می‌کند. انتشار شکل مؤثر شامل اشکال نقشه‌های ویژگی یا تنسورها در هر لایه از یک شبکهٔ عصبی عمیق است. انتشار شکل مؤثر، باید مطمئن شود که شکل تنسورها هنگام عبور از لایه‌های مختلف شبکه به طور مناسب تنظیم می‌شوند. این تنظیم می‌تواند شامل اعمال تغییراتی مانند تغییر اندازه، برش یا لایه‌بندی برای مطابقت با ابعاد مورد نیاز باشد[۱۵]؛ درنهایت FBNetV2 می‌تواند انتخاب‌های مختلف معماری مانند تعداد لایه‌ها، اندازهٔ هسته و اندازهٔ کانال را به‌عنوان متغیرهای پیوسته نشان دهد؛ سپس این متغیرها را می‌توان با استفاده از روش‌های کاهش گرادیان بهینه کرد.

OFA[4]: ایدهٔ کلیدی ONCE-FOR-ALL جداکردن معماری شبکه از عملیات سخت‌افزاری است. این کار با تعریف یک آبرشبکه شروع می‌شود که یک شبکهٔ عصبی بزرگ است که شامل تمام گزینه‌های ممکن معماری است. در طول فرایند آموزش، آبرشبکه بر روی یک مجموعه‌دادهٔ بزرگ آموزش داده می‌شود و معماری‌های بهینه را انتخاب می‌کند. این آموزش با ترکیبی از خطای سطح شبکه و سطح مسیر انجام می‌شود. از خطای سطح شبکه، برای بهبود عملکرد کلی شبکه استفاده می‌شود؛ درحالی‌که خطای سطح مسیر، باعث انتخاب مسیرهای بهینه در آبرشبکه می‌شود. هنگامی که آموزش کامل شد، آبرشبکه به زیرشبکه‌های کارآمد متعدد تبدیل می‌شود که برای بسترهای سخت‌افزاری مختلف می‌تواند استفاده شود[۱۷].

یکی از مسائلی که در روش‌های بالا به آن پرداخته نشد، بهینه‌سازی شبکه‌های عصبی برای سامانه‌های نهفتهٔ چندهسته‌ای است؛ به عبارت دیگر، در این روش‌ها، معماری شبکه‌ها به‌گونه‌ای طراحی نشده‌است که بتوانند به بهترین شکل از قابلیت‌های موازی پردازنده‌های چندهسته‌ای استفاده کنند؛ بنابراین، برای بهینه‌سازی عملکرد شبکه‌های عصبی بر روی سخت‌افزارهای چندهسته‌ای، نیاز است که معماری شبکه‌ها به‌گونه‌ای طراحی شود که بتوانند به بهترین شکل از قابلیت‌های موازی پردازنده‌های چندهسته‌ای بهره برده و از آن‌ها بهینه استفاده کنند. روش ارائه‌شده در این مقاله، به موازی‌سازی مدل در پردازنده‌های چندهسته‌ای، برای سرعت اجرای مدل در سامانه‌های نهفته می‌پردازد که دارای مزایای زیر است:

در سامانه‌های نهفته، سرعت اجرای مدل به زمانی اشاره دارد که یک مدل یا الگوریتم محاسباتی برای پردازش داده‌های ورودی و تولید خروجی در محدودیت‌های زمانی طول می‌کشد. هنگامی که یک مدل بر روی پردازنده‌های چندهسته‌ای اجرا می‌شود به‌عنوان پردازش موازی شناخته می‌شود، سرعت اجرا می‌تواند به‌طور قابل توجهی در مقایسه با اجرای مدل بر روی یک هسته بهبود یابد. به این دلیل است که هر هسته می‌تواند بخش متفاوتی از حجم کار مدل را هم‌زمان مدیریت کند؛ بنابراین زمان اجرای کلی را سرعت می‌بخشد. سرعت به‌دست‌آمده به‌وسیلهٔ پردازنده‌های چندهسته‌ای تا حد زیادی به ماهیت مدل و وظایفی که انجام می‌دهد بستگی دارد. در چنین مواردی، اگر بتوان حجم کار را بین هسته‌های موجود مساوی تقسیم کرد، می‌توان زمان اجرا را به نسبت کاهش داد؛ برای مثال، اگر یک مدل ده دقیقه طول می‌کشد تا روی یک هسته اجرا شود، اما اگر چهار هسته در دسترس باشد، اجرای آن به صورت موازی ممکن است زمان اجرا را به حدود ۲/۵ دقیقه کاهش دهد.

۳- پیش‌نیازها

در این بخش به پیش‌نیازهای روش ارائه‌شده می‌پردازیم:

۳-۱-سامانه‌های نهفته

سامانه‌های نهفته، سامانه‌هایی هستند که برای کاربردهای خاص متناسب با نیازهای سخت‌افزاری و نرم‌افزاری طراحی شده‌اند که بر حسب معمول در دستگاه‌هایی مورد استفاده قرار می‌گیرند که نیاز به پاسخ‌گویی سریع و مصرف انرژی کم دارند. سامانه‌های نهفته در طیف گسترده‌ای از کاربردهای مختلف مانند لوازم الکترونیکی مصرفی (تلفن‌های هوشمند، دوربین‌های دیجیتال)، خودرو (واحدهای کنترل موتور، سامانه‌های ترمز ضد قفل)، اتوماسیون صنعتی (رباتیک،

[1] Cross entropy
[2] Masking mechanism
[3] Effective shape propagation
[4] Once-for-All: Train One Network and Specialize it for Efficient Deployment

[DOI: 10.61186/jsdp.22.1.25]

کنترل فرایند)، دستگاه‌های پزشکی (دستگاه‌های قابل کاشت، سامانه‌های نظارت بر بیمار) و بسیاری دیگر مورد استفاده قرار می‌گیرند[۸].

با توجه به کاربردهای سامانه‌های نهفته، این سامانه‌ها ویژگی‌ها و محدودیت‌های دارند که عبارت‌اند از:

- عملیات‌های بی‌درنگ: بسیاری از سامانه‌های نهفته نیاز به پاسخ‌گویی بی‌درنگ دارند، در کاربردهایی که باید به رویدادهای خارجی در محدودیت‌های زمانی دقیق پاسخ دهند[۲۰].
- محدودیت‌های منابع: سامانه‌های نهفته با منابع محدودی مانند حافظه، قدرت پردازش و انرژی کار می‌کنند. بهینه‌سازی منابع برای برآوردن نیازهای سامانه در این محدودیت‌ها بسیار مهم است[۸].
- اندازهٔ کوچک و مصرف برق کم: سامانه‌های نهفته بیشتر فشرده و کم‌مصرف‌اند تا محدودیت‌های اندازه و توان را برآورده کنند[۲۱].

۳-۲-جست‌وجوی معماری عصبی در سامانه‌های نهفته

جست‌وجوی معماری عصبی روشی برای کشف خودکار معماری شبکهٔ عصبی بهینه برای یک کار معین است. علی‌رغم موفقیت قابل توجهی که تا به امروز به‌دست آورده‌است، استفاده از جست‌وجوی معماری عصبی برای مشکلات دنیای واقعی همچنان چالش‌های مهمی را به همراه دارد و به‌طور گسترده عملی نیست؛ درکل، معماری شبکهٔ عصبی کانولوشنی برای استقرار در بسترهای با منابع محدود، مانند اینترنت اشیا، تلفن همراه و سامانه‌های نهفته بسیار پیچیده‌است[۲۲]؛ بااین‌حال، علاقهٔ زیادی به استفاده از جست‌وجوی معماری عصبی در سامانه‌های نهفته وجود دارد، جایی که محدودیت‌های سخت‌افزاری نقش مهمی در تعیین معماری بهینهٔ شبکه ایفا می‌کنند.

یکی از چالش‌های اصلی استفاده از جست‌وجوی معماری عصبی در سامانه‌های نهفته، دسترسی محدود به منابع سخت‌افزاری مانند حافظه، قدرت پردازش و انرژی و اندازه و هزینه است. فضای جست‌وجو برای معماری‌های شبکه که می‌توانند بر روی سامانه‌های نهفته اجرا شوند، بر حسب معمول کمتر از فضای جست‌وجو برای دستگاه‌های محاسباتی همه‌منظوره است که کارایی روش‌های قبلی جست‌وجوی معماری عصبی را محدود می‌کند[۲۳].

برای مقابله با این چالش، پژوهش‌گران روش‌های سخت‌افزاری جست‌وجوی معماری عصبی را پیشنهاد کرده‌اند که محدودیت‌های سخت‌افزاری سامانه‌های نهفته را درنظر می‌گیرد. این روش‌ها می‌توانند به‌طور مؤثر فضای جست‌وجو را کاوش کنند و معماری‌های را پیدا کنند که برای بستر سخت‌افزاری هدف بهینه باشد[۸].

۳-۳-استقرار شبکهٔ عصبی در سامانهٔ نهفته

مراحل استقرار شبکهٔ عصبی در سامانهٔ نهفته در شکل (۲) خلاصه شده‌است. شبکهٔ عصبی ابتدا باید به‌اندازهٔ کافی با حجم زیادی از مجموعه‌داده‌ها آموزش داده شود. مجموعه‌داده‌های آموزشی تأثیر تعیین‌کننده‌ای بر دقت شبکهٔ عصبی دارد. آموزش و بهینه‌سازی شبکه به‌صورت کامل مرتبط با هم پردازش می‌شوند، به‌طوری‌که شبکهٔ عصبی پس از اعمال بهینه‌سازی‌ها که وظیفهٔ تنظیم دقیق وزن‌ها و کمینه‌کردن خطای داده‌ها را دارند، دوباره آموزش داده می‌شود. شبکه‌های عصبی داده‌های آموزشی را به‌صورت دسته‌ای آموزش می‌دهند که برای انجام این کار به سامانه‌های محاسباتی با کارایی بالا[1] (HPC) متکی‌اند. راه‌حل‌های سخت‌افزاری متعددی برای تسریع محاسبات شبکه‌های عصبی پیشنهاد شده‌اند[۳]، مانند:

۱) شتاب‌دهی مبتنی بر FPGA[2]: FPGA را می‌توان برای تسریع در اجرای الگوریتم‌های جست‌وجوی معماری عصبی که قابلیت بالایی در موازی‌سازی و سفارشی‌سازی دارند استفاده کرد[۲۴].

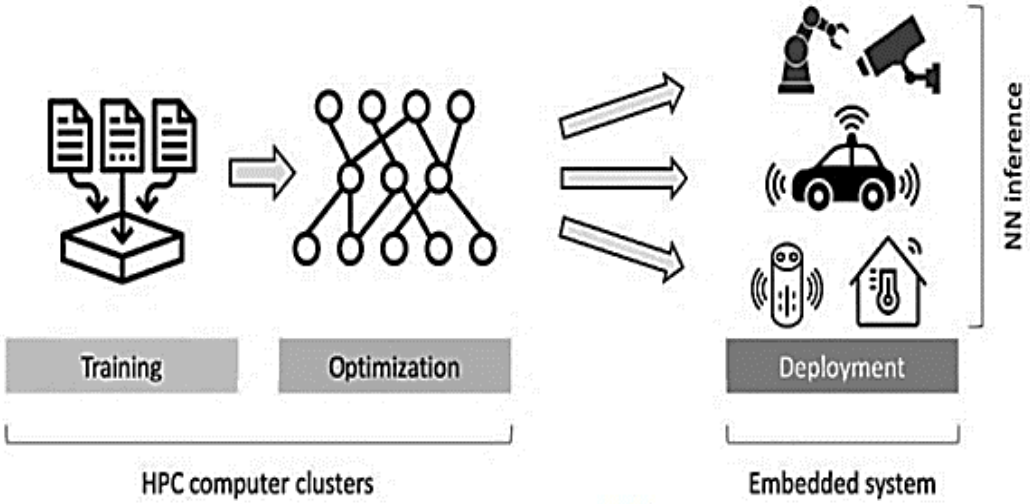
۲) راه‌حل‌های مبتنی بر ASIC[3]: ASIC را می‌توان با منابع سخت‌افزاری اختصاصی و بهینه‌سازی‌شده برای جست‌وجوی معماری شبکه‌های طراحی کرد[۲۵].

۳) شتاب‌دهی مبتنی بر GPU: GPU به‌طورمعمول برای کارهای یادگیری عمیق استفاده می‌شوند و برای سرعت‌بخشیدن به الگوریتم‌های جست‌وجوی معماری عصبی از قابلیت‌های پردازش موازی آن‌ها استفاده می‌شود.

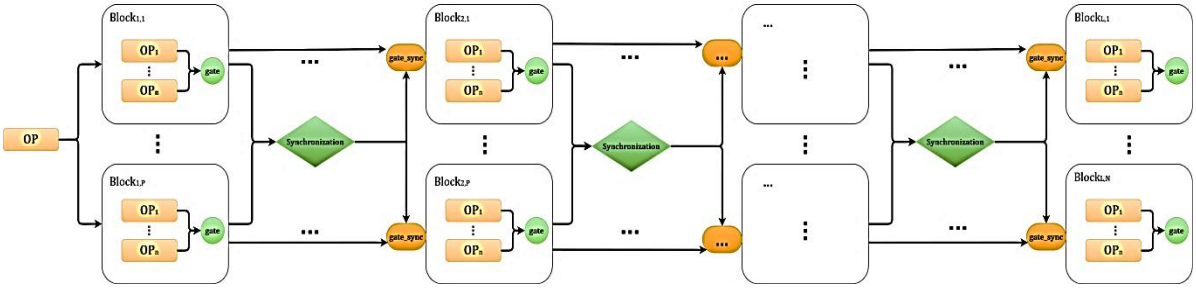
GPUها از این جنبه برترند؛ بنابراین آموزش شبکهٔ عصبی کمابیش همیشه با استفاده از GPUهای با کارایی بالا در مقیاس‌های بزرگ انجام می‌شود. پس از آموزش و بهینه‌سازی شبکهٔ عصبی، برای استفاده در سامانهٔ نهفته مستقر می‌شود. این کار را الگوریتم استنتاج در سامانهٔ نهفته انجام می‌دهد؛ مانند بازرسی محصولات در یک خط مونتاژ، ردیابی خودروها در جاده و غیره. پیاده‌سازی استنتاج‌های سریع (پیش‌بینی‌ها) شبکهٔ عصبی، مجموعه‌ای از چالش‌ها را بر سخت‌افزار نهفته تحمیل می‌کند. عملیات در سامانهٔ نهفته به‌طو معمول به صورت بی‌درنگ هستند؛ زیرا اجرای استنتاج باید در بازه‌های زمانی مشخص اجرا شود. استنتاج‌های سریع نیاز به قابلیت‌های محاسباتی بالا با تأخیر پایین دارد؛ بااین‌حال، بالابردن نقطهٔ کار (ولتاژ و فرکانس ساعت) واحدهای پردازش ممکن است منجر به تصمیمات متناقض در یک سامانهٔ نهفته شود که اغلب با باتری کار می‌کند؛ بنابراین، محدودیت‌های فیزیکی عوامل مهمی هستند که باید درنظر گرفته شوند؛ زیرا آن‌ها عملیات و عملکرد سامانهٔ نهفته را محدود می‌کنند.

[1] High-performance computing
[2] Field-programmable gate arrays
[3] Application Specific Integrated Circuits

[DOI: 10.61186/jsdp.22.1.25]



(شکل-۱): استقرار شبکهٔ عصبی در سامانهٔ نهفته[۸].
(Figure-1): Deployment of a neural network in an embedded system [8].



(شکل-۲): معماری روش پیشنهادی.
(Figure-2): Proposed Architecture Approach.

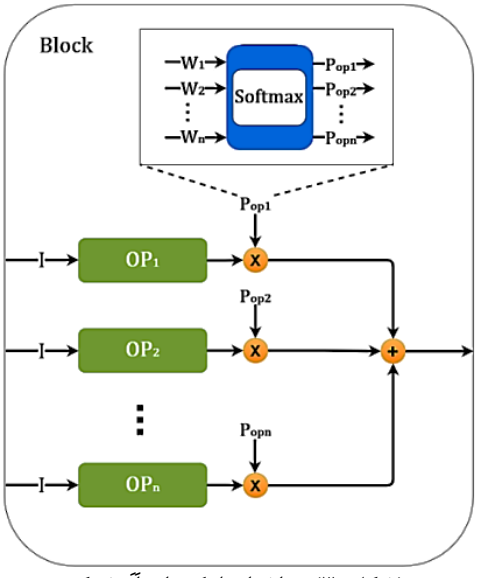
برای حل این مسئله روش‌هایی پیشنهاد شده‌است: استفاده از روش‌های بهینه‌سازی مختلف مانند: ۱) هرس [۲۶]، ۲) فشرده‌سازی مدل[۲۷]، ۳) کوانتیزاسیون[۲۸] و غیره است. انواع روش‌های بهینه‌سازی تلاش می‌کنند تا افزونگی را در محاسبات شبکه‌های عصبی کاهش دهند، ۴) انجام استنتاج مستقیم بر روی دستگاه‌های لبه مانند تلفن‌های هوشمند، دستگاه‌های اینترنت اشیا یا سامانه‌های نهفته، زمان لازم برای انتقال داده‌ها بین حس‌گرها و سرور راه دور را به کمترین حد می‌رساند. این امکان پردازش سریع‌تر داده‌ها و تولید سریع‌تر استنتاج می‌شود[۸].

۴- روش پیشنهادی

در روش پیشنهادی یک آبَرشبکه با مسیرهای موازی ایجاد می‌شود که مناسب اجرا بر روی پردازنده‌های چندهسته‌ای باشد. این روش بر اساس کاهش گرادیان است. به‌دلیل استفاده از روش کاهش گرادیان تنها دو بار فرایند آموزش انجام می‌شود و درنهایت ساختار نهایی شبکه به‌دست می‌آید. عملیات انجام‌شده در این مسیرها می‌تواند انواع محاسبات استفاده‌شده در شبکه‌های عصبی مانند انواع لایه‌های کانولوشن و لایهٔ به‌طور کامل متصل باشد. برای انتخاب عملیات‌های مناسب در هر بلوک از مسیر، از تعدادی متغیر تصمیم‌گیری استفاده می‌شود؛ درواقع در هر بلوک چندین عملیات وجود دارند که خروجی هر یک از آن‌ها در یک متغیر تصمیم‌گیری خاص ضرب می‌شود؛ سپس با هم جمع می‌شوند. با تغییر این متغیرها می‌توان تأثیر هر یک از این عملیات‌ها را کم یا زیاد کرد و درنهایت پس از پایان آموزش شبکه تنها یک عملیات که متغیر تصمیم‌گیری آن مقدار بیشتری دارد به‌عنوان عملیات نهایی در این بلوک انتخاب می‌شود.

برای استفادهٔ مسیرهای موازی از نتایج میانی هم‌دیگر و افزایش دقت شبکه، علاوه‌بر تصمیم‌گیری در مورد عملیات انجام‌شده در هر بلوک، در مورد نقاط همگام‌سازی نیز تصمیم‌گیری می‌شود؛ البته باید توجه کرد که همگام‌سازی بین چند مسیر (ریسمان) سربار زمانی دارد؛ درنتیجه نقاط همگام‌سازی تنها باید در زمانی صورت گیرد که درحقیقت باعث بهبود دقت شبکه شود؛ به همین دلیل است که در روش پیشنهادی در مورد وجود یا عدم وجود نقاط همگام‌سازی نیز تصمیم‌گیری انجام می‌شود. شکل (۳) معماری کلی شبکهٔ عصبی درنظر گرفته‌شده در این مقاله را نشان می‌دهد. در ادامه جزئیات انتخاب نوع محاسبات در هر بلوک و سپس نحوهٔ انتخاب نقاط همگام‌سازی بیان می‌شود.

[DOI: 10.61186/jsdp.22.1.25]



(شکل-۳): ساختار بلوک‌های آبرشبکه.

(Figure-3): The structure of SuperNet blocks.

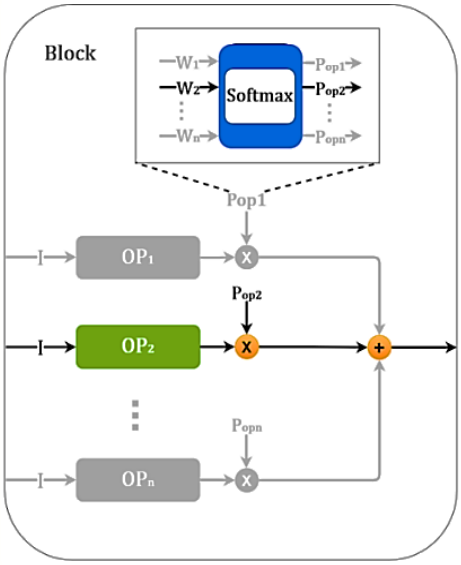
انتخاب نوع محاسبات در هر بلوک: عملیات قابل انتخاب در یک بلوک را به‌صورت OP_1، OP_2، OP_3 ... و OP_n نمایش می‌دهیم، OPها می‌توانند لایه‌های کانولوشن معمولی و کانولوشن قابل تفکیک با اندازهٔ هسته‌های مختلف باشند که در داخل بلوک آبرشبکهٔ روش پیشنهادی قرار می‌گیرند؛ مانند شکل (۴). برای اینکه بتوان یکی از این عملیات‌ها را انتخاب کرد، خروجی هر یک از این عملیات‌ها در یک مقدار (P_{op}) ضرب می‌شود. این مقدار بر اساس متغیر تصمیم‌گیری متناظر (w) و با استفاده از تابع Softmax به‌دست می‌آید؛ در واقع تابع Softmax مقداری در بازه [0,1] به هر یک از P_{op}ها تخصیص می‌دهد و مجموع مقادیر P_{op}ها برابر یک خواهد بود. رابطهٔ (۱) نحوهٔ محاسبهٔ هر P_{op} را نشان می‌دهد:

$$\forall\, i \in [1,N]: P_{op_i} = \frac{e^{w_i}}{\sum_{j=1}^{N} e^{w_j}} \qquad (۱)$$

پس از محاسبهٔ P_{op}ها، این مقادیر در خروجی عملیات‌ها ضرب شده و سپس به هم جمع می‌شوند. رابطهٔ (۲) نحوهٔ محاسبهٔ خروجی بلوک را نشان می‌دهد:

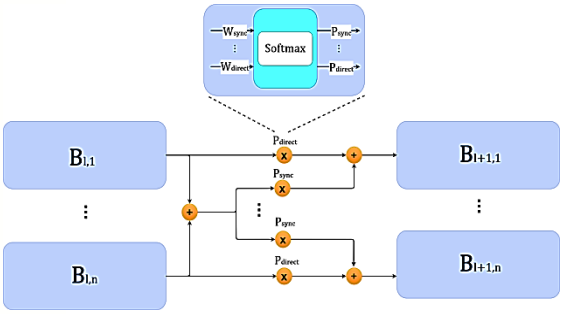
$$O = \sum_{i=1}^{N} OP_i(I) \times P_{op_i} \qquad (۲)$$

در این رابطه I ورودی و O خروجی بلوک است. بعد از این که متغیرهای تصمیم‌گیری پس از آموزش شبکه مقداردهی شد، عملیاتی که بیشترین P_{op} را دارد انتخاب شده و سایر عملیات‌ها از شبکه هرس می‌شوند، شکل (۵). شایان ذکر است که عملیات Softmax تنها در گام آموزش شبکه استفاده می‌شوند و پس از آموزش و مشخص‌شدن عملیات منتخب و همچنین نقاط همگام‌سازی حذف می‌شوند؛ درنتیجه در گام آزمایش و استقرار مدل بر روی سامانهٔ هدف این عملیات دیگر وجود ندارند و سربار در سامانهٔ نهایی ایجاد نمی‌کنند.



(شکل-۴): هرس عملیات‌هایی که کمترین P_{op} را در بلوک دارند.

(Figure-4): Pruning the operations that have the in the block. lowest P_{op}



(شکل-۵): ساختار نقاط همگام‌سازی در آبرشبکه.

(Figure-5): The synchronization points structure in the SuperNet.

نحوهٔ انتخاب نقاط همگام‌سازی: برای سادگی توضیحات این بخش، بلوک‌های ورودی به یک نقطهٔ همگام‌سازی (احتمالی) با $B_{l,i}$ و بلوک‌های خروجی از یک نقطهٔ همگام‌سازی با $B_{l+1,i}$ نمایش داده می‌شود، شکل (۶). برای مشخص‌کردن اینکه یک نقطهٔ همگام‌سازی وجود داشته باشد یا نه از دو متغیر تصمیم‌گیری استفاده می‌شود و مانند بخش پیشین این دو متغیر به‌عنوان ورودی به تابع Softmax داده و مقادیر خروجی در مسیر مستقیم و مسیر همگام‌سازی ضرب می‌شوند. مقادیر متغیرهای تصمیم‌گیری در زمان آموزش شبکه مشخص و هر کدام که مقدار بیشتری داشت مسیر مربوطه باقی‌مانده و مسیر دیگر هرس می‌شود.

به عبارت دیگر اگر متغیر تصمیم‌گیری مسیر همگام‌سازی مقدار بیشتری داشته باشد، نقطهٔ همگام‌سازی در شبکهٔ نهایی وجود خواهد داشت و در غیر

[DOI: 10.61186/jsdp.22.1.25]

این صورت هرس خواهد شد و بلوک‌های پیشین هر مسیر به صورت مستقیم به بلوک‌های بعدی همان مسیر متصل خواهند شد.

رابطهٔ (۳) نحوهٔ محاسبهٔ خروجی نقاط همگام‌سازی را نشان می‌دهد:

$$\forall\, i \in [1,N]: O_i = \left(s \times P_{l,sync} + B_{l,i}(I) \times P_{l,direct}\right) \quad (۳)$$

در این رابطه s نشان‌دهندهٔ حاصل اتصال خروجی تمام مسیرها در نقطهٔ همگام‌سازی مورد نظر است. برای عملیات اتصال می‌توان از عمل‌گرهای مختلفی مانند جمع و یا کانولوشن ۱×۱ استفاده کرد. در صورت استفاده از عمل‌گر جمع، s به صورت رابطهٔ (۴) محاسبه می‌شود:

$$s = \sum_{i=1}^{N} B_{l,i}(I) \quad (۴)$$

همچنین مجموع $P_{l,sync}$ و $P_{l,direct}$ نیز چون خروجی رابطهٔ softmax هستند برابر یک است.

۴-۱-تابع ضرر

برای اینکه الگوریتم آموزش شبکه را مجبور کنیم تا علاوه‌بر دقت شبکه به کاهش محاسبات شبکه نیز تمایل داشته باشد، از تابع ضرر چندبخشی استفاده می‌شود. بخشی از تابع ضرر مربوط به دقت شبکه است و مانند شبکه‌های طبقه‌بندی دیگر با آنتروپی متقابل پیاده‌سازی می‌شود. رابطهٔ (۵) رابطهٔ آنتروپی متقابل را نشان می‌دهد.

$$loss_{cross\_entropy} = -\sum P(x) \times \log(Q(x)) \quad (۵)$$

در رابطهٔ (۵) $P(x)$، توزیع احتمال طبقه‌ها یا نتایج درست را نشان می‌دهد. در وظایف طبقه‌بندی این توزیع یک احتمال به هر طبقه یا نتیجهٔ ممکن اختصاص می‌دهد. $Q(x)$ نیز توزیع احتمال پیش‌بینی‌شدهٔ طبقات یا نتایج را نشان می‌دهد. این توزیع یک احتمال پیش‌بینی‌شده را به هر طبقه یا نتیجهٔ ممکن اختصاص می‌دهد. بخش دیگر تابع ضرر مربوط به هزینهٔ محاسباتی شبکه است. برای محاسبهٔ هزینهٔ محاسباتی شبکه یا به عبارت دقیق‌تر تعداد عملیات ممیز شناور، هزینهٔ محاسباتی تمام عملیات انتخاب‌شده را با هم جمع می‌کنیم؛ همان‌طورکه در بخش‌های پیشین توضیح داده شد، عملیات انتخاب‌شده از روی مقدار متغیرهای تصمیم‌گیری مربوطه مشخص می‌شود. رابطهٔ (۶) هزینهٔ محاسباتی شبکه را به‌دست می‌آورد:

$$loss_{ops} = Norm\left(\sum_{l=1}^{L}\sum_{p=1}^{P}\sum_{n=1}^{N} P_{op_{l,p,n}} \times C_{l,p,n}\right) \quad (۶)$$

در رابطهٔ (۶) $P_{op_{l,p,n}}$، احتمال انتخاب عملیات n-ام در مسیر p-ام و در سطح l-ام $(op_{l,p,n})$ را نشان می‌دهد که نحوهٔ محاسبهٔ آن در رابطهٔ (۱) توضیح داده شد، $C_{l,p,n}$ نیز هزینهٔ محاسباتی عملیات‌ها را نشان می‌دهد.

تعداد ضرب‌های یک لایهٔ کانولوشنی را می‌توان به‌وسیلهٔ رابطهٔ (۷) محاسبه کرد:

$$Cmul_{conv}(k, W_o, H_o, D_i, D_o) = k^2 D_i W_o H_o D_o \quad (۷)$$

در این رابطه k نشان‌دهندهٔ ارتفاع یا عرض هسته، H_o و W_o به ترتیب نشان‌دهندهٔ ارتفاع و عرض خروجی و D_i و D_o به‌ترتیب نشان‌دهندهٔ تعداد کانال‌های ورودی و خروجی‌اند. تعداد جمع‌های انجام‌شده در لایهٔ کانولوشنی نیز با رابطهٔ (۸) محاسبه می‌شود.

$$Cadd_{conv}(k, W_o, H_o, D_i, D_o) = (k^2 D_i - 1) W_o H_o D_o \quad (۸)$$

در مورد لایه‌های کامل متصل نیز برای محاسبهٔ تعداد ضرب‌ها و جمع‌ها به‌ترتیب از روابط (۹) و (۱۰) استفاده شده‌است.

$$Cmul_{FC}(I, O) = O \times I \quad (۹)$$

$$Cadd_{FC}(I, O) = O \times (I - 1) \quad (۱۰)$$

در این روابط I و O به ترتیب بعد ورودی و خروجی را نشان می‌دهد. بخش بعدی تابع ضرر مربوط به نقاط همگام‌سازی است. هزینهٔ هر یک از نقاط همگام‌سازی نیز از روی متغیر تصمیم‌گیری مربوطه به‌دست می‌آید. در این بخش نیز مقدار ضرر به‌دست‌آمده نرمالیزه می‌شود. رابطهٔ (۱۱) نحوهٔ محاسبهٔ هزینهٔ نقاط همگام‌سازی را نشان می‌دهد.

$$loss_{sync} = \sum_{l=1}^{L-1} P_{Sync_l} \times C_{sync} \quad (۱۱)$$

در رابطهٔ (۱۱) P_{Sync_l}، احتمال این را که نقطهٔ همگام‌سازی l-ام انتخاب شود، نشان می‌دهد. C_{sync} نیز مقدار هزینهٔ سربار زمانی نقاط همگام‌سازی را نشان می‌دهد. مقدار C_{sync} نیز به شرط مساوی‌بودن پردازش مسیرها، بر اساس مدت زمان لازم برای محاسبهٔ حاصل ترکیب خروجی مسیرها قابل محاسبه است. پس از محاسبهٔ سه بخش ذکرشده، این مقادیر با هم جمع شده و تابع ضرر کلی به‌دست می‌آید. رابطهٔ (۱۲) تابع ضرر کلی را نشان می‌دهد:

$$loss_{total} = \alpha \times loss_{cross\ entropy} + \beta \times \left(loss_{ops} + loss_{syncs}\right) \quad (۱۲)$$

در رابطهٔ (۱۲) α و β به‌ترتیب وزن تأثیر تابع ضرر پیش‌بینی صحیح طبقهٔ دادهٔ ورودی و وزن تأثیر تابع ضرر مقدار محاسبات انجام‌شده در لایه‌ها و در نقاط همگام‌سازی هستند. با تغییر این وزن‌ها می‌توان تأثیر تابع ضرر مربوطه را در تابع ضرر کلی تغییر داد و درنتیجه نقطهٔ تعادل مناسب بین بار محاسباتی و دقت شبکه را به‌دست آورد.

[DOI: 10.61186/jsdp.22.1.25]

(جدول-۱): ساختار آبَر شبکه.
(Table-1): SuperNet Structure.

مسیر ۱	مسیر ۲
کانولوشن ۵×۵ (۸ کانال خروجی)	
کانولوشن ۵x۵ (۸ کانال خروجی)	کانولوشن ۵x۵ (۸ کانال خروجی)
کانولوشن ۳x۳ (۸ کانال خروجی)	کانولوشن ۳x۳ (۸ کانال خروجی)
نقطهٔ همگام‌سازی	
کانولوشن ۵x۵ (۱۶ کانال خروجی)	کانولوشن ۵x۵ (۱۶ کانال خروجی)
کانولوشن ۳x۳ (۱۶ کانال خروجی)	کانولوشن ۳x۳ (۱۶ کانال خروجی)
نقطهٔ همگام‌سازی	
کانولوشن ۵x۵ (۳۲ کانال خروجی)	کانولوشن ۵x۵ (۳۲ کانال خروجی)
کانولوشن ۳x۳ (۳۲ کانال خروجی)	کانولوشن ۳x۳ (۳۲ کانال خروجی)
نقطهٔ همگام‌سازی	
کانولوشن ۵x۵ (۶۴ کانال خروجی)	کانولوشن ۵x۵ (۶۴ کانال خروجی)
کانولوشن ۳x۳ (۶۴ کانال خروجی)	کانولوشن ۳x۳ (۶۴ کانال خروجی)
نقطهٔ همگام‌سازی	
پویش میانگین	
لایهٔ کامل متصل	
لایهٔ کامل متصل	

۵- آزمایش‌ها

تمام آزمایش‌ها بر روی مجموعه‌دادهٔ CIFAR-10 [۲۹] انجام شده‌است. مجموعه‌دادهٔ CIFAR-10 مجموعه‌ای از تصاویر است که بر حسب معمول برای وظایف تشخیص اشیا در پژوهش‌های بینایی رایانه‌ای استفاده می‌شود. این مجموعه‌داده شامل شصت‌هزار تصویر رنگی ۳۲×۳۲ در ده طبقه با شش‌هزار تصویر در هر طبقه است. طبقه‌ها عبارت‌اند از: هواپیما، اتومبیل، پرنده، گربه، آهو، سگ، قورباغه، اسب، کشتی و کامیون. مجموعه‌داده به پنجاه‌هزار تصویر آموزشی و ده‌هزار تصویر آزمایشی تقسیم شده‌است. تمام روش‌ها با کاربست پایتورچ [۳۰] پیاده‌سازی شده‌است. برای انجام آزمایش‌ها از کارت گرافیکی RTX 3090 و پردازندهٔ Cori7 نسل یازده با مقدار حافظهٔ ۳۲ گیگابایت استفاده شده‌است. فضای جست‌وجوی آزمایش (ساختار آبَرشبکه) در جدول (۱) نمایش داده شده‌است. در آزمایش‌های انجام‌شده در این مقاله از عملیات‌های قابل انتخاب کانولوشن با هسته‌های ۳×۳ و ۵×۵ استفاده شده‌است. لایه‌های آخر شبکه نیز همانند شبکه‌های متداول طبقه‌بندی تصاویر از لایه‌های کامل متصل تشکیل شده‌است؛ اگرچه در آزمایش‌هایی که در این بخش انجام می‌شود تنها تعدادی عملیات و یک نوع شبکه (طبقه‌بندی) بررسی شده‌است، روش ارائه‌شده محدود به عملیات خاصی مانند کانولوشن و یا لایه‌های کامل متصل نیست و از آن می‌توان در انواع شبکه‌ها مانند بخش‌بندی و تشخیص اشیا نیز استفاده کرد؛ برای مثال در شبکه‌های تشخیص اشیا می‌توان شبکهٔ استخراج‌کنندهٔ ویژگی‌ها را با استفاده از روش ارائه‌شده جست‌وجو کرده و به صورت درشت دانه موازی‌سازی کرد.

همان‌طورکه در شکل (۶) مشخص است پس از هر لایه محاسبات امکان انتخاب نقاط همگام‌سازی در آبَرشبکه درنظر گرفته شده‌است. برای همگام‌سازی از یک لایهٔ کانولوشنی ۱x۱ استفاده شده‌است که خروجی مسیرها را به‌عنوان ورودی گرفته و تعداد کانال‌ها را به تعداد مناسب تبدیل می‌کند (عرض و طول نقشهٔ ویژگی‌ها عوض نمی‌شود). الگوریتم بهینه‌سازی که برای آموزش شبکه در این مقاله استفاده شد، الگوریتم کاهش گرادیان تصادفی (SGD) است. برای وزن‌های اصلی شبکه از نرخ یادگیری ۰.۰۰۰۱ و برای متغیرهای تصمیم‌گیری (انتخاب نقاط همگام‌سازی و عملیات در هر لایه) از نرخ یادگیری ۰.۱ استفاده می‌شود. برای بررسی عملکرد ایده‌های ارائه‌شده آزمایش‌هاس زیر انجام شده‌است:

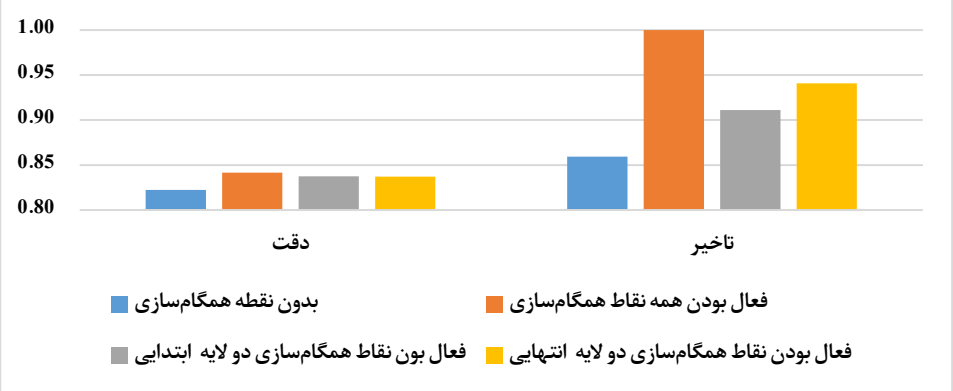
۱) بررسی اثر نقاط همگام‌سازی در افزایش دقت و سربار شبکه
۲) بررسی اثر وزن‌های توابع ضرر در جابه‌جایی نقطهٔ تعادل دقت و محاسبات شبکه

در ادامه نتایج آزمایش‌های گزارش می‌شود.

۵-۱-بررسی اثر نقاط همگام‌سازی در دقت و سربار محاسباتی شبکه

برای آزمایش اثر نقاط همگام‌سازی پیکربندی‌های مختلفی از حالت‌های ممکن از نظر دقت و تأخیر آزمایش شده‌است. پیکربندی‌های آزمایش‌شده بدین صورت‌اند:

[DOI: 10.61186/jsdp.22.1.25]



(شکل-۷): اثر نقاط همگام‌سازی بر دقت و تأخیر شبکه.

(Figure-7): The impact of synchronization points on network accuracy and latency

۱. بدون نقطهٔ همگام‌سازی
۲. فعال‌بودن تمام نقاط همگام‌سازی
۳. فعال‌بودن دو نقطهٔ ابتدایی همگام‌سازی
٤. فعال‌بودن دو نقطهٔ انتهایی همگام‌سازی

برای انجام آزمایش‌های تأخیر، مدل‌ها به فرمت onnx تبدیل شدند تا از بهینه‌سازی موجود در این فرمت استفاده شود. آزمایش سرعت با ورودی با اندازهٔ batch=1 انجام شده‌است. برای افزایش دقت مقایسه، هر آزمایش صد بار تکرار شده‌است و میانگین نتایج گزارش شده‌است. برای ارزیابی دقت شبکه‌ها نیز هر آزمایش پنج بار آموزش دیده‌است و دقت متوسط گزارش شده‌است. در هر لایه یک مسیر با کانولوشن ۳x۳ و مسیر دیگر با کانولوشن ۵x۵ درنظر گرفته شده‌است. شکل (۷) تأخیر و دقت شبکه‌ها را نشان می‌دهد. همان‌طورکه در این شکل مشخص است، شبکه با فعال‌بودن تمام نقاط همگام‌سازی به بهترین دقت در بین شبکه‌های ارزیابی‌شده رسیده‌است؛ همچنین نمودار نشان می‌دهد هنگام فعال‌بودن تمام نقاط همگام‌سازی بیشترین تأخیر شبکه اتفاق افتاده‌است. کمترین تأخیر و کمترین دقت مربوط به حالتی است که هیچ نقطهٔ همگام‌سازی فعال نبوده‌است. سایر حالت‌ها از نظر دقت و تأخیر در بین این دو حالت قرار دارند. نتایج این آزمایش نشان می‌دهد که وجود نقاط همگام‌سازی باعث افزایش دقت شبکه می‌شود؛ البته تأخیر شبکه نیز زیاد می‌شود که برای ایجاد تعادل می‌توان حالت‌های میانی را در نظر گرفت.

۵-۲-بررسی اثر وزن‌های توابع ضرر در جابه‌جایی نقطه تعادل دقت و محاسبات شبکه

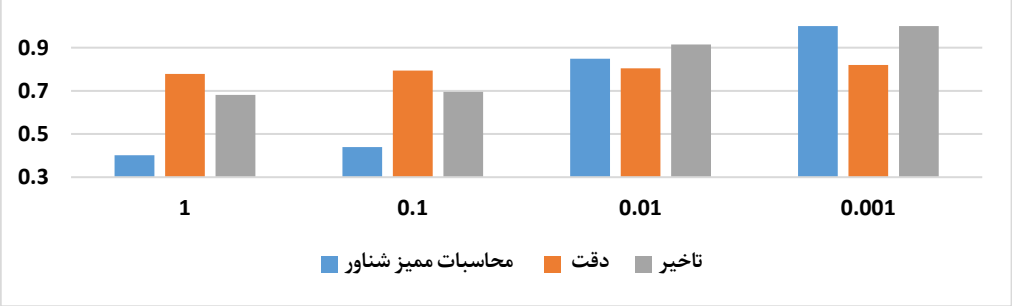
در این آزمایش آبَر شبکه با مقادیر مختلف وزن‌های تأثیر تابع ضرر (α, β) آموزش می‌بینند. در همه آزمایش‌ها مقدار $\alpha = 0.28$ در نظر گرفته می‌شود. در لایه‌های آبَر شبکه در هر مسیر، دو نوع عملیات کانولوشن ۳x۳ و ۵x۵ قابل انتخاب هستند. پس از آموزش با ضرایب تأثیر مختلف پیکربندی شبکه به‌دست‌آمده در هر حالت در جدول (۲) نشان داده شده‌است؛ همچنین دقت، تأخیر و مقدار محاسبات ممیز شناور شبکه پس‌آموزش مجدد هر یک از شبکه‌ها در شکل (۸) نشان داده شده‌است. نتایج تأخیر و تعداد محاسبات ممیز شناور به‌صورت عادی نمایش داده شده‌اند؛ همان‌طورکه در جدول مشخص است، با افزایش ضریب β شبکه ترغیب می‌شود تا از عملیات با محاسبات کمتری استفاده کند. با کاهش β تأثیر هزینهٔ محاسباتی در تابع ضرر کاهش می‌یابد و فرایند آموزش ترغیب می‌شود تا دقت شبکه را بالا ببرد. با انتخاب مقدار مناسب برای β می‌توان نقطهٔ تعادل مناسب بین دقت و سرعت شبکه را برقرار کرد.

(جدول-۲): اثر ضریب تأثیر هزینهٔ محاسباتی شبکه بر ساختار شبکه

(Table-2): The effect of the impact factor of the computational cost of the network.

ضریب تأثیر هزینهٔ محاسباتی شبکه	ساختار به‌دست‌آمده			
	لایهٔ ۱	لایهٔ ۲	لایهٔ ۳	لایهٔ ۴
۱	Conv 3x3 - Conv 3x3 No sync	Conv 3x3 - Conv 3x3 No sync	Conv 3x3 - Conv 3x3 No sync	Conv 3x3 - Conv 3x3 No sync
۰.۱	Conv 3x3 - Conv 3x3 No sync	Conv 3x3 - Conv 3x3 No sync	Conv 3x3 - Conv 3x3 No sync	Conv 3x3 - Conv 5x5 No sync
۰.۰۱	Conv 3x3 - Conv 5x5 No sync	Conv 5x5 - Conv 5x5 No sync	Conv 5x5 - Conv 5x5 No sync	Conv 5x5 - Conv 5x5 concat
۰.۰۰۱	Conv 5x5 - Conv 5x5 No sync	Conv 5x5 - Conv 5x5 concat	Conv 5x5 - Conv 5x5 No sync	Conv 5x5 - Conv 5x5 concat

[DOI: 10.61186/jsdp.22.1.25]



(شکل-۸): **اثر ضریب تأثیر هزینهٔ محاسباتی شبکه** بر دقت، تأخیر و تعداد محاسبات ممیز شناور.

(Figure-8): The effect of the impact factor of network computational cost on accuracy, latency, and the number of floating-point operations.

۶- نتیجه‌گیری

در این مقاله روشی جهت موازی‌سازی معماری شبکه عصبی ارائه شد که مناسب اجرا بر روی پردازنده‌های چندهسته‌ای است. در این روش که بر اساس کاهش گرادیان است، یک آبرشبکهٔ موازی از بلوک‌های محاسباتی ایجاد می‌شود؛ سپس با آموزش متغیرهای تصمیم‌گیری (نوع بلوک و نقاط همگام‌سازی) هم‌زمان با وزن‌های اصلی شبکه، زیرشبکه مناسب انتخاب می‌شود.

در این روش به‌دلیل استفاده از روش کاهش گرادیان تنها دو بار فرایند آموزش انجام می‌شود؛ درنهایت ساختار نهایی شبکه به‌دست می‌آید و درنتیجه نسبت به سایر روش‌های مبتنی بر جست‌وجوی تکاملی و یادگیری تقویتی، زمان جست‌وجو بسیار کمتر خواهد شد. نکتهٔ قابل توجه در روش ارائه‌شده این است که در تابع ضرر مقدار هزینهٔ محاسباتی شبکه نیز درنظر گرفته می‌شود و درنتیجه شبکهٔ نهایی علاوه‌بر دقت مناسب، هزینهٔ محاسباتی کمی نیز خواهد داشت؛ همچنین با تنظیم پارامترها در تابع ضرر می‌توان به تعادل مناسبی بین دقت و سرعت شبکه دست یافت.

نتایج به‌دست‌آمده از آزمایش‌ها، درستی روش پیشنهادی را تأیید کرده‌است. روش ارائه‌شده به‌دلیل نوع معماری شبکه به سادگی قابلیت موازی‌سازی و اجرای واقعی بر روی پردازنده‌های چندهسته را داشته و نسبت به روش‌های متعارف که به طور معمول تک مسیره‌اند قابلیت افزایش سرعت بیشتری در پردازنده‌های چند هسته ایجاد می‌کند. در کارهای آتی می‌توان با اجرا بر سامانه‌های نهفته که دسترسی به پردازندهٔ گرافیکی میسر نیست، مقدار تأخیر و حافظهٔ مصرفی را به‌صورت دقیق‌تر بررسی کرد؛ همچنین روش ارائه‌شده را می‌توان در فضای حالت گسترده‌تری از انواع عمل‌گرها و روش‌های همگام‌سازی بررسی کرد.

۷-مراجع 7-References

[1] T. Elsken, J. H.Metzen, and F. Hutter, “Neural Architecture Search: A Survey.” arXiv, Apr. 26, 2019.

[2] M. Wistuba, A. Rawat, and T. Pedapati, “A Survey on Neural Architecture Search.” arXiv, Jun.18,2019.

[3] H. Benmeziane, K. E. Maghraoui, H. Ouarnoughi, S. Niar, M. Wistuba, and N. Wang, “A Comprehensive Survey on Hardware-Aware Neural Architecture Search.” arXiv,Jan.22,2021.

[4] X. He, K. Zhao, and X. Chu, “AutoML: A survey of the state-of-the-art,” *Knowl.-Based Syst.*, vol. 212, p. 106622, Jan. 2021.

[5] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, “Regularized Evolution for Image Classifier Architecture Search.” arXiv, Feb. 16, 2019.

[6] B. Zoph and Q. V. Le, “Neural Architecture Search with Reinforcement Learning.” arXiv, Feb.15,2017.

[7] H. Liu, K. Simonyan, and Y. Yang, “DARTS: Differentiable Architecture Search.” arXiv, Apr.23,2019.

[8] W. J. Song, “Chapter Two - Hardware accelerator systems for embedded systems,” in *Advances in Computers*, vol. 122, S. Kim and G. C. Deka, Eds., in Hardware Accelerator Systems for Artificial Intelligence and Machine Learning, vol. 122. , Elsevier, 2021, pp. 23–49.

[9] H. Park and S. Kim, “Chapter Three - Hardware accelerator systems for artificial intelligence and machine learning,” in *Advances in Computers*, vol. 122, S. Kim and G. C. Deka, Eds., in Hardware Accelerator Systems for Artificial Intelligence and Machine Learning, vol. 122. , Elsevier, 2021, pp. 51–95.

[10] A. G. Howard *et al.*, “MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications.” arXiv, Apr. 16, 2017.

[11] X. Zhang, X. Zhou, M. Lin, and J. Sun, “ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices.” arXiv, Dec. 07, 2017.

[12] M. Tan and Q. V. Le, “EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks.” arXiv, Sep. 11, 2020.

[13] H. Cai, L. Zhu, and S. Han, “ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware.” arXiv, Feb. 22, 2019.

[14] B. Wu *et al.*, “FBNet: Hardware-Aware Efficient ConvNet Design via Differentiable

[DOI: 10.61186/jsdp.22.1.25]

Neural Architecture Search." arXiv, May 24, 2019.
[15] A. Wan *et al.*, "FBNetV2: Differentiable Neural Architecture Search for Spatial and Channel Dimensions." arXiv, Apr. 12, 2020.
[16] X. Chen, L. Xie, J. Wu, and Q. Tian, "Progressive Differentiable Architecture Search: Bridging the Depth Gap Between Search and Evaluation," presented at the Proceedings of the IEEE/CVF International Conference on Computer Vision, 2019, pp. 1294–1303. Accessed: Aug. 30, 2023. [Online]. Available: https://openaccess.thecvf.com/content_ICCV_2019/html/Chen_Progressive_Differentiable_Architecture_Search_Bridging_the_Depth_Gap_Between_Search_ICCV_2019_paper.html
[17] H. Cai, C. Gan, T. Wang, Z. Zhang, and S. Han, "Once-for-All: Train One Network and Specialize it for Efficient Deployment." arXiv, Apr.29,2020.
[18] "Automated deep learning architecture design using differentiable architecture search (DARTS)." Accessed: Aug. 21, 2023. [Online]. Available: https://mountainscholar.org/handle/10217/199856
[19] D. Stamoulis *et al.*, "Single-Path NAS: Designing Hardware-Efficient ConvNets in less than 4 Hours." arXiv, Apr. 05, 2019.
[20] C. Li *et al.*, "HW-NAS-Bench:Hardware-Aware Neural Architecture Search Benchmark." arXiv, Mar. 18, 2021.
[21] L. L. Zhang, Y. Yang, Y. Jiang, W. Zhu, and Y. Liu, "Fast Hardware-Aware Neural Architecture Search," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, Seattle, WA, USA: IEEE, Jun. 2020, pp. 2959–2967.
[22] K. T. Chitty-Venkata and A. K. Somani, "Neural Architecture Search Survey: A Hardware Perspective," *ACM Comput. Surv.*, vol. 55, no. 4, p. 78:1-78:36, Nov. 2022.
[23] W. Roth *et al.*, "Resource-Efficient Neural Networks for Embedded Systems." arXiv, Dec. 09, 2022.
[24] "How to Boost Compute Performance with FPGA-Based Accelerators - element14 Community." Accessed: Aug. 22, 2023. [Online].Available:https://community.element14.com/technologies/fpga-group/w/documents/5003/how-to-boost-compute-performance-with-fpga-based-accelerators.
[25] "Ultimate Guide: ASIC (Application Specific Integrated Circuit)," AnySilicon. Accessed: Aug. 22,2023. [Online]. Available: https://anysilicon.com/ultimate-guide-asic-application-specific-integrated-circuit/
[26] S. Han, H. Mao, and W. J. Dally, "Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding." arXiv, Feb. 15, 2016.
[27] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both Weights and Connections for Efficient Neural Networks." arXiv, Oct. 30, 2015.
[28] B. Jacob *et al.*, "Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, Jun. 2018, pp. 2704–2713.
[29] "CIFAR-10 and CIFAR-100 datasets." Accessed: Sep. 14, 2023. [Online]. Available: https://www.cs.toronto.edu/~kriz/cifar.html
[30] "PyTorch." Accessed: Sep. 16, 2023. [Online]. Available: https://www.pytorch.org

سهیل رستاری تحصیلات کارشناسی مهندسی کامپیوتر گرایش نرم‌افزار را در دانشگاه شهید چمران رشت و کارشناسی‌ارشد مهندسی کامپیوتر گرایش نرم‌افزار را در دانشگاه صنعتی قم به پایان رساند. زمینه‌های پژوهشی وی شبکه‌های عصبی عمیق و سامانه‌های نهفته است.
نشانی رایانامهٔ ایشان عبارت است از:
rastari.s@qut.ac.ir

مرتضی محجل کفشدوز تحصیلات خود را در رشتهٔ مهندسی کامپیوتر در مقاطع کارشناسی، کارشناسی‌ارشد و دکترا را در دانشگاه صنعتی شریف به پایان رسانده و از سال ۹۶ عضو هیئت علمی دانشگاه صنعتی قم است. زمینه‌های پژوهشی ایشان شبکه‌های عصبی عمیق و سامانه‌های نهفته است.
نشانی رایانامهٔ ایشان عبارت است از:
mohajjel@qut.ac.ir

محبوبه شمسی تحصیلات خود را در مقطع کارشناسی در رشتهٔ ریاضی کاربردی در کامپیوتر در دانشگاه اصفهان و مقطع کارشناسی‌ارشد را در رشتهٔ مهندسی کامپیوتر در دانشگاه اصفهان به پایان رساند؛ همچنین دکترای نرم‌افزار خود را در دانشگاه مالزی (UTM) در سال ۸۹ به پایان رسانده‌است و از سال ۹۱ عضو هیئت علمی دانشگاه صنعتی قم در گروه کامپیوتر است. زمینه‌های پژوهشی ایشان پردازش تصویر و طراحی سامانه‌های امنیتی بیومتریکی و سامانه‌های بیوانفورماتیکی است.
نشانی رایانامهٔ ایشان عبارت است از:
Shamsi@qut.ac.ir

[DOI: 10.61186/jsdp.22.1.25]