گزارش پروژه پایانی پیاده‌سازی مقاله:

Approximation-based Fault Tolerance in Image Processing Applications

Matteo Biasielli, Cristiana Bolchini, Senior Member, IEEE, Luca Cassano, Member, IEEE, Andrea Mazzeo, Antonio Miele, Senior Member, IEEE,

چکیده مقاله مرجع

تحمل خطا[1] در پردازش تصویر به دو دلیل: ۱) طبیعت افزونگی[2] در عکس‌ها و ۲) امکان استفاده از خروجی نهایی توسط مصرف‌کننده حتی اگر عکس دارای خطاهایی در اطلاعات باشد، کاربرد بالقوه و غیرقابل‌اغماضی دارد. استفاده از تکنیک کلاسیک DWC[3] در اینجا خیلی سخت‌گیرانه خواهد بود چرا که اگر دو تصویر کاملاً با هم یکی نباشند آن‌ها را رد می‌کند که این مطابق میل ما نیست.

در این مقاله روشی به‌مراتب آسان‌گیرانه‌تر از DWC ارائه شده : ۱) در این روش یکی از کپی‌های دقیق جایگزین نمونه تقریبی خواهد شد و ۲)به‌جایی استفاده از گروه‌بندی دارای خطا و بدون خطا دودسته بندی قابل‌استفاده و بلااستفاده ارائه شده که این امر با استفاده از ابزار چک کننده CNN[4] پیاده‌سازی شده. برای میزان کردن طرح ارائه شده، متدولوژی[5] خاصی معرفی شده که هم زمان اجرا و هم قابلیت تشخیص خطا[6] در سیستم سخت‌افزاری را بهبود می‌بخشد. نتایج ارائه شده برای کاربرد این روشی که مقاله داده به شرح ذیل است: ۱) کاهش متوسط زمان اجرا بالغ بر ۳۰٪ به نسبت روش DWC و ۲) کمتر از ۴٪ اشتباه در قراردادن تصاویر در دسته غیرقابل‌استفاده‌ها.

[1] Fault tolerance
[2] Reduandancy
[3] Duplicant With Comparission
[4] Convolutional neural network
[5] Methodology
[6] Fault detection

۱) مقدمه

پردازش تصویر به طور وسیعی برای عملکردهای ادراکی چون دستیار کنترل خودمختار در سفینه فضایی و رباط راننده خودمختار و ... ساخته شده و مورد کاربری قرار گرفته اگر مبنا را انجام چنین عملیات بدانیم چنین سیستم‌های محاسباتی باید در دسته سیستم‌هایی ایمن برای انجام مأموریت حیاتی با مکانیزمی که تحمل خطا را دارد و سطح مورد انتظار از اطمینان را برآورده می‌کند قرار گیرند، حتی سیستم‌هایی که امروزه درحال‌توسعه هستند برای چنین کاربردهایی سخت‌افزارهایی گران‌قیمت اما به طور معمول کندتر را تولید می‌کنند که حتی بیت به بیت خروجی را دوباره و سه باره چک می‌کنند که خطایی وجود نداشته باشد، تمامی این سیستم‌هایی که به آن‌ها اشاره شد دارای هزینه بالا، زمان اجرای بالا و توان مصرفی بالا خواهند بود.

برای کاهش هزینه سربار در پیاده‌سازی چنین سیستم‌های سخت‌افزاری باید دو نکته را در نظر بگیریم: ۱) در سیستم‌هایی که پیش‌تر به آن‌ها اشاره کردیم هر دوی کاربردهای حیاتی و غیرحیاتی در کنار هم وجود دارند.

۲) کاربردهایی که مربوط به پردازش تصویر می‌شوند به دلیل ذات منعطفی که پردازش تصویر در برابر خطا دارد می‌توانند تا حدی در برابر خطاهایی چشم‌پوشی کنند و سیستم آن‌ها را تحمل کند.

در یک سمت در کاربردهای این‌چنینی ورودی‌های سیستم به‌صورت مکرر دارای نویز[۷] یا کوانتیزه شدگی[۸] هستند که همین باعث وجود اشکال[۹] در خروجی است، در سمت دیگر کاربری مثل انسان وجود دارد که می‌تواند خروجی را حتی با وجود تغییر جزئی و یا نویز استفاده کند، وجود چنین طبیعت منعطف در برابر خطایی که اشاره شد در دهه گذشته به‌صورت گسترده‌ای مورداستفاده قرار گرفته تا بین کارایی سیستم و انرژی که سیستم مصرف می‌کند و کیفیتی که خروجی آن دارد یک هماهنگی و تعادلی وجود داشته باشد.

در چند سال اخیر ذات تحمل‌پذیری در برابر خطا که سیستم‌های پردازش تصویر دارند مورد کاربرد قرار گرفته تا روش نوینی برای کشف خطا به وجود بیاید، این شیوهٔ نوین سعی دارد تا تمرکز را روی سیستمی که خروجی‌های آن در دودسته قابل‌استفاده و غیرقابل‌استفاده است و قابلیت این را دارد که حدس بزند خروجی قابل‌استفاده برای کارهای سیستم هست یا خیر قرار دهد که این کلاس‌بندی را CNN انجام می‌دهد، پیاده‌سازی سخت‌افزاری بر اساس این است که تشخیص دهد خروجی با وجود خطاهایی که دارد آیا می‌تواند انتظارات ما برآورده سازد و اینکه روی خروجی نهایی سیستم این خطاها تأثیری دارد یا خیر همچنین سخت‌افزاری که پیاده شده پایپ لاین[۱۰] های گذشته را به بلوک‌های کنترلی [۱۱] تبدیل کرده که به‌صورت تقریبی کار چک‌کردن

[7] Noise
[8] Quantized
[9] Error
[10] Pipeline
[11] Control blocks

را برای هر بلاک انجام می‌دهد، پروپوزالی که معرفی شد شیوه‌ای را ارائه می‌کند که زمان بهبود[12] کمتری دارد؛ اما هزینه سربار آن به‌اندازه شیوه DWC هست.

۲) مدل سیستم

مدل طراحی شده این مقاله بهبودیافته مدل DWC است، مدل اصلی خود یک کنترل تحمل‌پذیر اشکال برای سیستم‌های پردازش تصویر است، اما در این مدل حتی بیت‌ها هم باید با هم برابر باشد و خروجی باید بدون هیچ‌گونه اشکال یا نویزی باشد؛ اما این مقاله با استفاده از دو رویکرد: ۱) استفاده از CNN برای دسته‌بندی عکس‌ها در دو گروه قابل‌استفاده و غیرقابل‌استفاده (که البته مانند CWP سربار ۲ برابری دارد؛ ولی به طرز شگفت‌آوری زمان اجرای سخت‌افزاری کاهش می‌یابد) و ۲) پیاده‌سازی سخت‌افزاری که دارای بلوک‌های کنترلی شامل کنترل‌کننده‌های هوشمند ((SCs[13] است که با هرکدام از پایپ لاین‌هایی که تشکیل شده از فیلتر برای پردازش تصویر جفت شده و به طور تقریبی تصویر ورودی را چک می‌کند.

شکل ۱ شمای کلی از مدل ارائه شده توسط مقاله را نشان می‌دهد، در این شکل عکس ورودی به خط لوله شبکه عصبی (CNN) تحت عنوان input_image وارد شبکه عصبی می‌شود تا از لایه‌های مختلف عبور کند و در نهایت خروجی شبکه عصبی تحت عنوان output_image ، طبق الگوریتم کلاس‌بندی[14] که شبکه عصبی از قبل برای آن آموزش‌دیده عکس خروجی را در یکی از کلاس‌هایی که برای آن تعریف شده قرار دهد، اما موازی با هر لایه در CNN یک بلوک کنترلی (cb) قرار گرفته که در ادامه کارکرد آن را توضیح خواهیم داد:

۲.۱. نحوه کارکرد بلوک کنترلی:

شکل ۲ شمای کلی از یک بلوک کنترلی را نشان می‌دهد، هر بلوک کنترلی از یک ماژول ورودی کوچک، دو کپی از فیلتر (fi) استفاده شده در لایه موازی با آن که به‌صورت موازی کار می‌کنند به نام‌های $f_i^{\wedge}$ ، یک TRC و SC تشکیل شده است. در حالت کلی کارکرد یک cb شامل مراحل زیر است :

۱) یک ورودی به نام in_img را دریافت می‌کند که دقیقاً همان ورودی لایه موازی با آن است، ورودی دیگر آن out_img است که خروجی fi لایه متناظر با آن است و ورودی سوم آن هم همان عکس اصلی و ورودی خام اولیه شبکه CNN است .

۲) ورودی in_img با یک فاکتور کاهش مقیاس[15] di کوچک می‌شود (برای هر دو بعد تصویر اعمال می‌شود)

[12] Recovery time
[13] Smart Checkers
[14] Classification
[15] Downscaling factor

۳) سپس دو فیلتر تکرار شده $\hat{f_i}$ به صورت موازی روی تصویر کوچک شده اجرا می‌شوند. اجرا بر روی تصاویر کوچک‌شده، تکنیک محاسباتی تقریبی در سطح کاربرد[۱۶] را شامل می‌شود که توسط طرح پیشنهادی برای دستیابی به تحمل خطای سبک وزن[۱۷] مورد استفاده قرار می‌گیرد.

۴) شروع اجرای TRC روی تصاویر خروجی دو فیلتر تکرار شده، یعنی ctr_img و ctrl_img 2، به این معنا که دو عکس به‌صورت بیت به بیت با هم مقایسه می‌شوند و خروجی به‌صورت ok/fail است ، اگر خروجی ok باشد؛ یعنی هر دو عکس با فیلترهای تقریبی به‌صورت سالم هستند و مرحله بعدی یعنی sc که در cb قرار گرفته شروع به کار می‌کند، اگر خروجی به‌صورت fail باشد؛ یعنی یکی از فیلترهای تقریبی fault خورده و لایه متناظر موازی با آن مجدداً اجرا می‌شود.

۵) این مرحله درصورتی‌که خروجی در مرحله قبل ok باشد اجرا می‌شود ، sc جزء اصلی طرح تحمل خطا پیشنهادی است که مغز سیستم را تشکیل می‌دهد و مسئول تصمیم‌گیری در مورد حذف تصاویر و اجرای مجدد (D) یا نه (/D) است. همان‌طور که قبلاً بحث شد، یک SC برای هر CB نمونه‌سازی می‌شود. هر sc سه ورودی زیر را می‌گیرد:

i) تصویر خروجی فیلتر اصلی (fi) به نام out_img،

ii) تصویر خروجی تولید شده توسط یکی از دو $\hat{f_i}$ ، (ctr_img) را دریافت می‌کند ،

iii) تصویر ورودی خط لوله (in_img) و

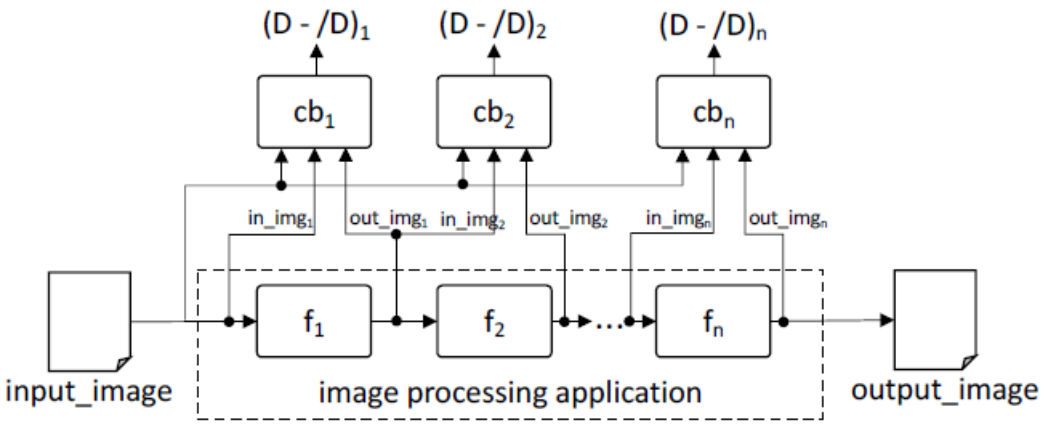
iv) پیام ok/fail تولید شده توسط TRC در مرحله قبل.

بر اساس سه تصویر ورودی که SC با آن تغذیه می‌شود ، SC تصمیم می‌گیرد که آیا imgi را کنار بگذارد و مرحله یکم خط لوله را مجدداً اجرا کند یا خیر.

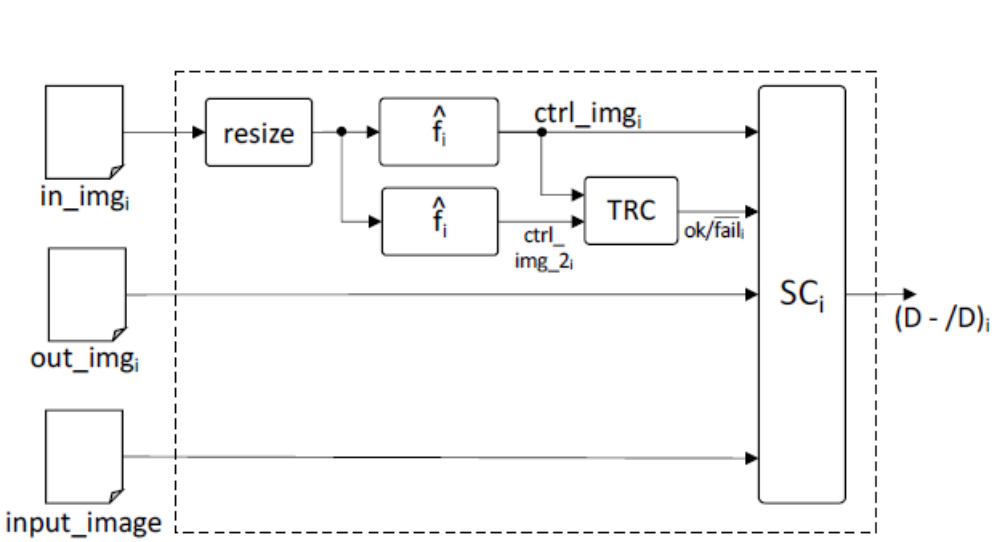
ازآنجایی‌که out_imgi و ctrl_imgi نیز در اجرای بدون خطا یکسان نیستند، SC باید بین تفاوت‌های ناشی از تقریب و تفاوت‌های ناشی از وقوع خطا (در بالای تقریب) تمایز قائل شود. علاوه بر این، هنگامی که وضعیت دوم رخ می‌دهد، SC باید تأثیر آن را بر خروجی نهایی پیش‌بینی کند، به‌گونه‌ای که تصمیم بگیرد عکس مذکور قابل‌استفاده هست یا نه و در این حالت خروجی مرحله یکم خط لوله کنار گذاشته شده و یک فیلتر مجدداً اجرا می‌شود. برای انجام این کار، هر SC از یک CNN بهره‌برداری می‌کند که قابلیت استفاده پیش‌بینی‌شده (U) یا نه (/U) را اعلام می‌کند. به طور خلاصه، هنگامی که سیگنال ok/fail در مرحله ۴ خطا باشد، خروجی SC به

[16] Application-level approximate computing technique
[17] Lightweight fault tolerance

طور خودکار روی /D تنظیم می‌شود: در این سناریو در واقع یک خطا در یکی از دو فیلتر عکس‌های تقریبی رخ‌داده است.



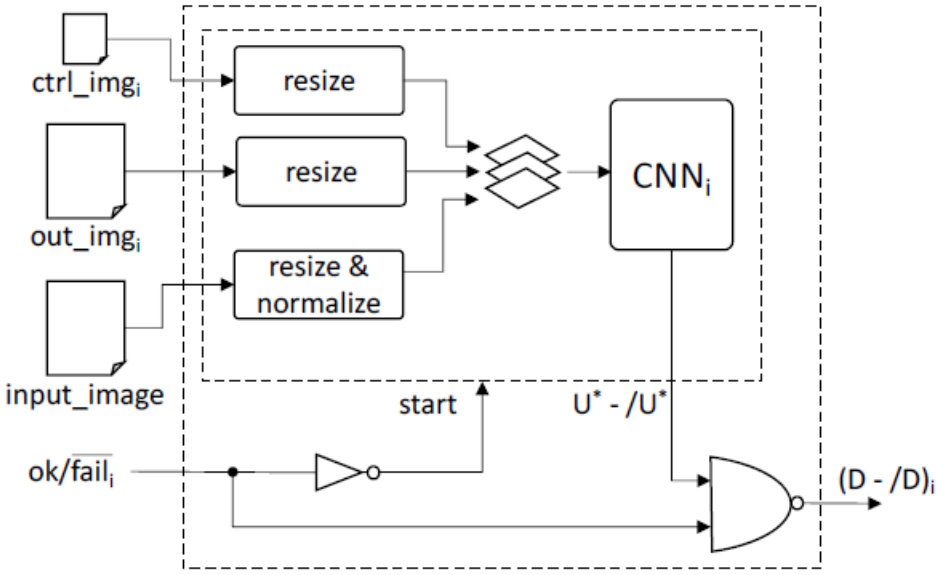
شکل ۱: یک شمای کلی از طرح تحمل خطا پیشنهادی مقاله.



شکل ۲: ساختار داخلی cb متناظر با لایه i از خط لوله شبکه عصبی

همان‌طور که در شکل ۳ می‌بینید، تصاویر imgi، ctrl_imgi و in_img قبل از ارسال به CNN کوچک می‌شوند. در واقع imgi و in_img بزرگ‌تر از ctrl_img هستند، درحالی‌که ورودی CNN طراحی‌شده باید یک ماتریس

سه‌بعدی باشد، جایی که سه تصویر روی‌هم چیده می‌شوند؛ بنابراین، imgi و in_img برای مطابقت با اندازه ctrl_imgi کوچک می‌شوند. علاوه بر این، تصاویر imgi و in_img را می‌توان به طور معقولی کاهش داد تا زمان اجرای CNN را کاهش دهد و درعین‌حال دقت دسته‌بندی آن را حفظ کند. بنابراین، imgi و in_img را می‌توان با ضریب dcnn i تغییر اندازه داد، و ctrl_imgi را با ضریب dcnni کاهش مقیاس داد، بنابراین سه تصویر کوچک‌تر با همان اندازه به دست می‌آیند که باید روی‌هم چیده شوند. در نهایت، اگر ctrl_imgi و imgi نقشه‌های ویژگی[۱۸] یا نقشه‌های حرارتی[۱۹] (به‌جای تصاویر) با مقادیری در محدوده متفاوت از شدت رنگ پیکسل‌های تصویر ورودی باشند، تصویر ورودی نرمال می‌شود. این یک گام ضروری برای ایجاد ورودی‌های یکنواخت‌تر و سهولت آموزش CNN و همچنین افزایش قابلیت پیش‌بینی مدل است. خروجی CNN یک مقدار احتمال ucnn است که قابلیت استفاده خروجی نهایی برنامه را تخمین می‌زند. این مقدار در نهایت با یک آستانه Uth مقایسه می‌شود و طبقه‌بندی نهایی U یا U/ را ایجاد می‌کند.



شکل ۳: ساختار داخلی SC متناظر با لایه i از خط لوله شبکه عصبی

[18] Feature maps
[19] Heat maps

۳) ایرادات مدل

۱. این شیوه برای تشخیص خطا کاملاً وابسته به شبکه عصبی که استفاده می‌کند هست درصورتی‌که اگر خود شبکه (مثل وزن‌ها فیلترهایی که استفاده می‌شود) در هنگام اجرا (و نه در هنگام training) دچار خطا شود (مثلاً SEU[20] بخورد) ، تشخیص قابل‌استفاده بودن به مشکل خواهد خورد.

۲. همان‌طور که در خود مقاله هم اشاره شده این شیوه نمی‌تواند برای کارهای حیاطی پیاده‌سازی شود چرا که با اینکه زمان اجرا را تا حد خوبی کاهش داده؛ ولی منفی‌های اشتباه (false negatives) آن در حدود کمتر از ۴٪ است (که این تا حد زیادی به آستانه‌ای CNN به نام U_{th} بستگی دارد) که در مقایسه با DWC (با صفر درصد false negative) که سربار یکسانی با این شیوه دارد یک ایراد محسوب می‌شود.

۳. این شیوه تنها روی دو مطالعه موردی بررسی شده درصورتی‌که می‌توانست روی پایگاه‌داده‌های مختلف با انواع موضوعات برای پردازش تصویر بررسی شده و خروجی را مورد ارزیابی قرار گیرد تا بتواند بر اساس استدلال قوی‌تری ادعای خود که مبنی بر کاربردی‌بودن این شیوه است را اثبات کند.

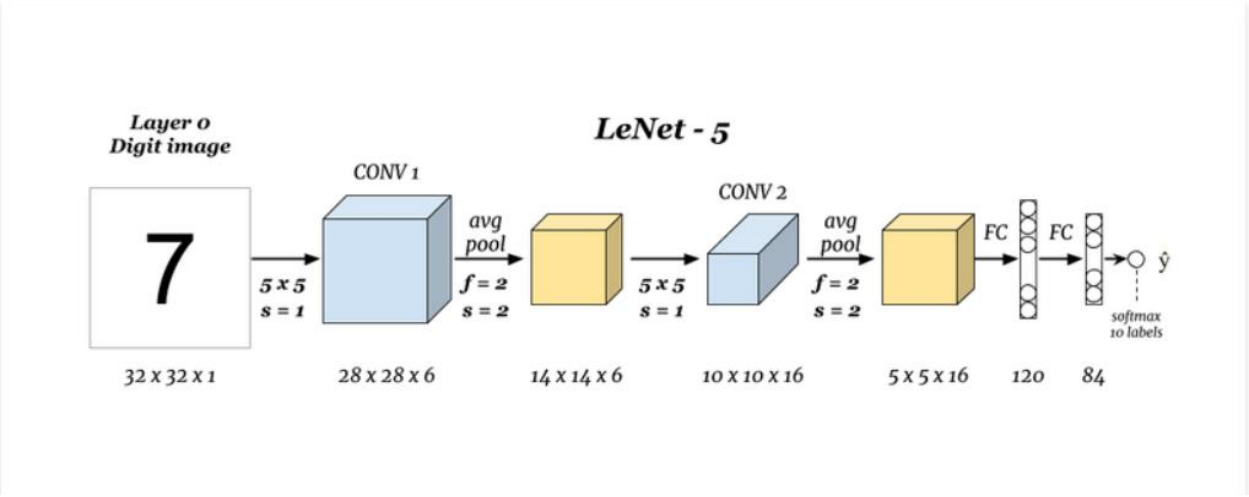
۴) روند پیاده‌سازی

در ابتدا توضیحی در مورد مکانیزم کلی سیستم معرفی شده در مقاله و نقشه کلی پیاده‌سازی آن توسط نویسنده این گزارش می‌پردازیم سپس به‌صورت جزئی تمامی مراحل پیاده‌سازی را به طور مفصل شرح می‌دهیم.

۴.۱. مکانیزم کلی سیستم:

همان‌طور که در بخش مدل سیستم هم توضیح داده شد یک خط لوله شبکه عصبی داریم که تصاویر را به‌صورت ورودی می‌گیرد و در نهایت تصمیم می‌گیرد که هر تصویر به چه کلاسی تعلق دارد، شبکه عصبی که نویسنده گزارش برای پیاده‌سازی انتخاب کرده Lenet-5 است که ساختار لایه‌های آن در شکل ۴ نشان‌داده‌شده:

[20] Single Event Upset



شکل ۴ : دیاگرامی از ساختار شبکه عصبی lenet-5

دلیل انتخاب شبکه lenet-5 وجود لایه‌های متعدد convolution و pooling و لایه fully connected در آن است که بیشترین تطابق را با شبکه عصبی که خود مقاله استفاده کرده (شبکه عصبی مورداستفاده مقاله نامی ندارد، صرفاً در مقاله گفته شده که از تعدادی لایه convolution و pooling و لایه fully connected در پیاده‌سازی استفاده شده)، دارد. لازم به ذکر است از هر شبکه عصبی که برای کلاس‌بندی می‌توان از آن استفاده کرد، می‌توانیم استفاده کنیم و همین نقطه قوت مقاله است که پیاده‌سازی را به یک شبکه عصبی خاص محدود نمی‌کند.

در شبکه عصبی ما (lenet-5) موازی با هر لایه دارای فیلتر (pooling و convolution) یک CB قرار می‌گیرد ، در پیاده‌سازی که نویسنده این گزارش کرده، برای اینکه کار در مقیاس پایین انجام شود و از ایجاد پیچیدگی بی‌مورد جلوگیری شود، CB برای لایه Convolution اول موازی با این لایه قرار گرفته و بعد از اجرای این لایه بلافاصله اجرا می‌شود تا متوجه شود که عکسی که خروجی لایه اول است سالم مانده باشد و برای لایه‌های بعدی قابل‌استفاده باشد و خروجی نهایی آن قابل‌تشخیص و درست باشد، بنابراین مهم‌ترین بخش پیاده‌سازی مربوط به پیاده‌سازی یک CB دقیق است که با بهینه‌ترین دسته‌بندی عکس‌ها در دودسته قابل‌استفاده و غیرقابل‌استفاده، کمترین سربار را به نسبت سیستمی با پیاده‌سازی DWC یا TMR داشته باشد و درست‌ترین نتیجه را بگیرد، برای پیاده‌سازی یک CB دقیق باید دو پارامتر Uth و d را که در ادامه در مورد آن توضیح خواهیم داد را به‌درستی انتخاب کنیم، برای همین شبکه‌ای که در CB قرار گرفته باید با Uth, و d های مختلف آموزش داده شود تا شبکه‌ای که بهترین ترکیب Uth و d را دارد برای مدل مذکور انتخاب کنیم.

۴.۲.تعریف حد آستانه U [۲۱]:

حد آستانه U یا به‌اختصار Uth یک عدد بین ۰ و ۱ است که به‌وسیله آن قابل‌استفاده بودن یا نبودن یک عکس توسط شبکه عصبی داخل CB تعریف می‌شود به این صورت که اگر عکسی که فیلتر تقریبی روی آن خورده (ctrl_img) را به‌صورت بیت به بیت با عکسی که فیلتر اصلی رو آن خورده (out_img) مقایسه کنیم و تعداد پیکسل‌هایی که اختلاف آن‌ها از یک حد آستانه (که مقاله ۲ آن را ۰.۰۵ فرض کرده) بیشتر است، بشماریم و نسبت این تعداد پیکسل‌ها به کل پیکسل‌های یک عکس از 1 – Uth بیشتر باشد آن عکس جزو دسته غیرقابل‌استفاده قرار می‌گیرد، در واقع، Uth ممکن است به‌عنوان درجه اطمینان از قابلیت استفاده ورودی‌هایی که CNN برای طبقه‌بندی تصویر به‌عنوان قابل‌استفاده نیاز دارد، تفسیر شود. به طور شهودی، هرچه Uth بزرگ‌تر باشد، تصاویری که توسط CNN پذیرفته می‌شوند کمتر است. این به نوبه خود، پتانسیل کاهش مقدار منفی‌های کاذب را دارد (تصاویر خراب غیرقابل‌استفاده که به‌اشتباه پذیرفته شده‌اند) اما مقدار مثبت کاذب را افزایش می‌دهد (تصاویر خراب قابل‌استفاده یا حتی تصاویر خراب نشده که CNN را به اجرای مجدد غیرضروری سوق می‌دهد). باتوجه‌به فضای راه‌حل ما CNN را با بهترین تعادل بین درصد منفی‌های اشتباه و صرفه‌جویی در زمان انتخاب می‌کنیم.

۴.۳.تعریف تقسیم‌کننده [۲۲](d):

میزان کوچک‌شدن تصویر در بخش تغییر اندازه[۲۳] در CB به این پارامتر بستگی دارد، در واقع این پارامتر میزان کوچک‌شدن تصاویر را هم در بخش اعمال فیلتر تقریبی(di) و هم در بخش تغییر سایز عکس در شبکه عصبی متعلق به CB (dcnn) تعیین می‌کند ،در واقع یکی از نکات مهم پیاده‌سازی مدل پیشنهادی مقاله انتخاب تکنیک‌های محاسبات تقریبی[۲۴] (AC) مناسب برای تولید فیلترهای تقریبی $f_i^{\wedge}$ برای فیلتر fi است. طراح کسی است که انتخاب را بر اساس تخصص و دانش در کل برنامه انجام می‌دهد. اگرچه روش شناسی کلی است و می توان از AC های مختلف استفاده کرد، اما برای سادگی در این کار، تکنیک کاهش مقیاس[۲۵] را اتخاذ می‌کنیم. در این زمینه، طراح مجموعه‌ای از مقادیر مناسب را برای ضریب کاهش مقیاس di انتخاب می‌کند، که ۱ کران پایین است، که با تقریب/کاهش مقیاس مطابقت ندارد. کران بالایی dmax i با آموزش شبکه عصبی درون CB مشخص می‌شود که یک فضای محاسباتی را بوجود می آورد که جلو تر در مورد آن توضیح خواهیم داد. به‌طور دقیق‌تر، برای شناسایی چنین کران بالایی، به طور مکرر ضریب مقیاس‌سازی ورودی‌های CNN را افزایش می‌دهیم، تا زمانی که دقت طبقه‌بندی به‌دست‌آمده نسبت به قبل از تغییر روند بالا و پایدار باشد. هنگامی که

[21] U threashold
[22] Devisor
[23] Resize
[24] Approximate computing
[25] Downscaling

dmax i مشخص شد، مقادیر در [۱; dmax i] بازه با انتخاب همه مقسوم‌کننده‌های ممکن اندازه‌های تصویر ورودی گنجانده می‌شود.

همان‌طور که گفتیم بعد از مقایسه دو عکس cntr_img1 و cntrl_img2 که خروجی فیلترهای تقریبی هستند اگر نتیجه مقایسه OK باشد شبکه عصبی پیاده شده داخل CB تصمیم می‌گیرد که عکس out_img که همان خروجی فیلتر اصلی است آیا قابل‌استفاده است یا خیر، این مقاله برای پیاده‌سازی شبکه عصبی خود از ابزاری به نام oracle استفاده کرده که کارکرد آن را به طور مفصل در ادامه توضیح خواهیم داد؛ اما باتوجه به اینکه ما برای پیاده‌سازی به این ابزار دسترسی نداشتیم از یک تکنیک دیگر استفاده کردیم که درواقع سعی کرده ایده اصلی oracle را پیاده کند و عملکردی مبتنی بر منطق ابزار oracle داشته باشد تا نتایج به‌دست‌آمده از آن درست باشد.

۴.۴.ابزار oracle:

یک برنامه کاربر نهایی[۲۶] ساده برای تقسیم‌بندی که عکس (احتمالاً معیوب) of و عکس همتای مربوطه og را می‌گیرد و درصورتی‌که فاصله منهتن (یا L1-norm) دو بردار خروجی $\|o_f - o_g\|$ کمتر از آستانه ۰.۰۵ باشد یک برچسب U (قابل‌استفاده) را به‌عکس ورودی نسبت می‌دهد و در غیر این صورت (فاصله منتهن[۲۷] از ۰.۰۵) بیشتر باشد برچسب /U (غیرقابل‌استفاده) را برمی‌گرداند.

به طور مثال در پیاده‌سازی تشخیص ساختمان در عکس‌های هوایی توسط مقاله برنامه مورد استفاده توسط مقاله (oracle) ساختمان‌ها را در تصاویر هوایی شناسایی می‌کند. این برنامه یک bitmap را می‌پذیرد و یک نقشه حرارتی با همان اندازه تولید می‌کند که در آن هر پیکسل ورودی در تصویر ورودی با احتمال تعلق پیکسل به یک ساختمان کدگذاری می‌شود. سپس جعبه‌های مرزی[۲۸] بر روی نقشه حرارتی ترسیم می‌شوند تا ساختمان‌های شناسایی‌شده برجسته شوند. برنامه کاربردی از چهار مرحله تشکیل شده است:

i) تیزکردن[۲۹] (S) یک پیچیدگی که یک هسته [۳۰]تیز کننده را اعمال می‌کند. این مرحله یک تصویر می‌گیرد و تصویر واضح شده مربوطه را تولید می‌کند.

[26] End user
[27] Manhattan
[28] Boundin -boxes
[29] Sharpening
[30] Kernel

ii) آستانه‌گذاری (T) یک طبقه‌بندی پیکسلی را با انجام یک مقایسه کانالی مقادیر[31] انجام می‌دهد.

iii) resharpe&convolution یک طبقه‌بندی را بر اساس تغییر شکل تصویر و سپس یک کانولوشن انجام می‌دهد. دو مرحله ذکر شده، تصویر شارپ شده را می‌گیرند و دو ماتریس از مقادیر احتمال را تولید می‌کنند.
iv) تجمع (A) یک ضرب پیکسلی بین خروجی‌های دو مرحله قبلی را انجام می‌دهد تا طبقه‌بندی‌های آنها را جمع کند. این مرحله دو ماتریس می‌گیرد و نقشه حرارتی خروجی را تولید می‌کند. به‌عنوان یک برنامه کاربر نهایی ساده برای شناسایی ساختمان، این مقاله oracle را پیاده‌سازی کرده که یک نقشه حرارتی (احتمالاً معیوب) داده می‌شود و اگر جعبه‌های مرزی در دو نقشه حرارتی حداقل برای ۸۵ درصد با هم همپوشانی داشته باشند، یک برچسب قابل‌استفاده تولید می‌کند.

تکنیکی که ما جایگزین ابزار oracle کردیم دقیقاً از منطق همین ابزار پیروی می‌کند؛ یعنی قرار است همان دو ورودی ctrl_img و out_img که به oracle می‌دادیم را با تکنیکی مشابه ابزار oracle مقایسه کنیم و اگر دو تصویر out_img و ctrl_img به‌اندازه Uth یا بیشتر با هم مطابقت داشتند عکس out_img برچسب قابل‌استفاده بگیرد ، برای مقایسه ctrl_img وout_img ما بیت به بیت این دو عکس را با هم مقایسه می‌کنیم اگر در هر دو پیکسل متناظر قدرمطلق اختلاف دو پیکسل از 0.05 بیشتر باشد یک واحد به شمارنده‌ای که در نظر گرفتیم اضافه می‌کنیم، در نهایت اگر نسبت مقدار شمارنده به تعداد کل پیکسل‌ها از Uth بیشتر باشد یعنی out_img قابل‌استفاده است و برچسب U به آن تعلق می‌گیرد.

۴.۵.فاز آموزش SC :

برای اینکه بهترین مقدار را برای Uth و d تعیین کنیم لازم است تا شبکه عصبی SC را بارها آموزش دهیم تا بهترین انتخاب را داشته باشیم، معیارهایی که مشخص می‌کند مقدار تنظیم شده برای Uth و d مناسب بوده یکی زمان اجرا و دیگری i) درصد تصاویر غیرقابل‌استفاده که به‌اشتباه توسط SC پذیرفته شده است (منفی‌های نادرست)، ii) میانگین زمان اجرای برنامه محافظت شده. اولی ازنقطه‌نظر قابلیت اطمینان بسیار مهم است، درحالی‌که دومی مزایای راه‌حل را اندازه‌گیری می‌کند. در پیاده‌سازی که نویسنده این گزارش انجام داده ۲ مقدار مختلف برای Uth و دو مقدار مختلف برای d در نظر گرفته و حالت‌های مختلف انتخاب Uth و d با هم که ۴ حالت می‌شود را در نظر گرفته و در نتیجه الگوریتم پیشنهادی برنامه را ۴ مرتبه به‌ازای همهٔ ترکیب‌های Uth و d اجرا کرده و در نهایت باتوجه‌به اینکه معیار اولی توسط مقاله مهم‌تر شمرده شده (دستیابی به کمترین میزان D/U/ یکی از ۴ حالت را که کمترین منفی نادرست) (D/U/ را نسب به بقیه داشته به‌عنوان بهترین ترکیب Uth و d انتخاب می‌کند.

[31] Channel -wise comparison

به‌طورکلی برای ارزیابی SC پیاده‌سازی شده عکس‌ها در ۴ دسته‌بندی زیر قرار می‌گیرند:

۱) D/U/ : آنهایی که توسط SC رد نشدند (نیاز به اجرای مجدد لایه متناظر نیست) ولی قابل‌استفاده نبودند (یک انسان یا ابزار oracle مورداستفاده مقاله نمی‌تواند عکس را با وجود نویزها کلاس‌بندی کند)
۲) D/U : آنهایی که SC غیرقابل‌استفاده تشخیص داده و در واقع هم غیرقابل‌استفاده بوده‌اند.
۳) DU/ : آنهایی که SC قابل‌استفاده تشخیص داده و در واقع هم قابل‌استفاده بوده‌اند.
۴) DU : آنهایی که SC غیرقابل‌استفاده تشخیص داده؛ ولی در واقع قابل‌استفاده بوده‌اند.

۴.۶.پیاده‌سازی کد سیستم به همراه جزئیات مدل:

در این بخش در مورد کدی که پیاده‌سازی کرده‌ایم توضیح خواهیم داد و به سایر جزئیات پیاده‌سازی مقاله که پیش‌تر به آن اشاره نکرده‌ایم در خلال مطالب خواهیم پرداخت.

پایگاه‌داده‌ای که برای پیاده‌سازی استفاده کرده‌ایم mnist است، پایگاه‌داده mnist یک پایگاه‌داده بزرگ از ارقام دست‌نویس است که معمولاً برای آموزش سیستم‌های مختلف پردازش تصویر استفاده می‌شود.

با استفاده از کد زیر داده‌های آموزش[۳۲] و برچسب‌های آن‌ها را از پایگاه‌داده mnist می‌گیریم و به ترتیب درون متغیرهای train_x و train_y و همچنین داده‌های آزمایش[۳۳] و برچسب‌های آن‌ها را از پایگاه‌داده mnist می‌گیریم و به ترتیب درون متغیرهای test_x و test_y می‌ریزیم.

```
from keras.datasets import mnist

(train_X, train_y), (test_X, test_y) = mnist.load_data()
```

سپس برچسب‌های عکس‌های پایگاه داده را با صدا زدن تابع convertToOneHotبه صورت one hot در می‌آوریم :

```
train_labs = convertToOneHot(train_labs)
```

معرفی تابع convertToOneHot : این تابع آرایه یک بعدی از برچسب‌ها را می‌گیرد و یک آرایه دوبعدی با مقادیر صفر به نام oneHotLabels ایجاد می‌کند ، سپس به ترتیب از خانه صفرم این آرایه به خانه متناظ با مقدار برچسب، مقدار ۱ را اختصاص می‌دهد:

```
def convertToOneHot(labels):
```

[32] Train
[33] Test

```
    oneHotLabels = np.zeros((labels.size, labels.max()+1))
    oneHotLabels[np.arange(labels.size), labels] = 1
    return oneHotLabels
```

حال کلاس Lenet را با نرخ یادگیری[34] دلخواه (۰.۰۱) صدا می‌زنیم و آن را درون متغیر my_CNN می‌ریزیم:

```
my_CNN = LeNet(lr)
```

معرفی کلاس Lenet : این کلاس شامل ۶ تابع است که به ترتیب آن‌ها را معرفی می‌کنیم:

تابع ___init_ : برای مقدار دهی به متغیرها از آن استفاده می‌شود، در بخشی از این کلاس برای مقدار دهی لایه‌های conv و fc از تابع xavier_init استفاده شده که در ادامه در مورد آن توضیح خواهیم داد:

```
class LeNet(object):
    #The network is like:
    #    conv1 -> pool1 -> conv2 -> pool2 -> fc1 -> relu -> fc2 -> relu -> softmax
    # l0     l1      l2      l3       l4    l5     l6    l7     l8       l9
    def __init__(self, lr=0.1):
        self.lr = lr
        # 6 convolution kernal, each has 1 * 5 * 5 size
        self.conv1 = xavier_init(6, 1, 5, 5)
        # the size for mean pool is 2 * 2, stride = 2
        self.pool1 = [2, 2]
        # 16 convolution kernal, each has 6 * 5 * 5 size
        self.conv2 = xavier_init(16, 6, 5, 5)
        # the size for mean pool is 2 * 2, stride = 2
        self.pool2 = [2, 2]
        # fully connected layer 256 -> 200
```

[34] Learning rate

```
        self.fc1 = xavier_init(256, 200, fc=True)
        # fully connected layer 200 -> 10
        self.fc2 = xavier_init(200, 10, fc=True)
```

تابع forward propagation که داده اولیه (input_data) را می‌گیرد و به ترتیب در لایه‌ها به سمت جلو آن را حرکت می‌دهد، به طور مثال self.l1 لایه convolution ۱ است که ورودی آن همان self.l0 (خروجی لایه صفر) است و پارامترهای مورد استفاده در convolution آن در تابع ___init_ تحت عنوان self.conv1 مقدار دهی شده:

```
    def forward_prop(self, input_data):
        self.l0 = np.expand_dims(input_data, axis=1) / 255   # (batch_sz, 1, 28, 28)
        self.l1 = self.convolution(self.l0, self.conv1)      # (batch_sz, 6, 24, 24)
        self.l2 = self.mean_pool(self.l1, self.pool1)        # (batch_sz, 6, 12, 12)
        self.l3 = self.convolution(self.l2, self.conv2)      # (batch_sz, 16, 8, 8)
        self.l4 = self.mean_pool(self.l3, self.pool2)        # (batch_sz, 16, 4, 4)
        self.l5 = self.fully_connect(self.l4, self.fc1)      # (batch_sz, 200)
        self.l6 = self.relu(self.l5)                         # (batch_sz, 200)
        self.l7 = self.fully_connect(self.l6, self.fc2)      # (batch_sz, 10)
        self.l8 = self.relu(self.l7)                         # (batch_sz, 10)
        self.l9 = self.softmax(self.l8)                      # (batch_sz, 10)
        return self.l9
```

تابع backward_prop که در مسیر لایه‌ها را از انتها به ابتدا حرکت می‌کند و عمل backpropagation را روی CNN انجام می‌دهد برای اینکه وزن‌ها را بروز رسانی کند، همانطور که در کد زیر می‌بینید پارامتر deriv که به تابع‌ها(مثلاً self.relu) پاس داده می‌شود مقدار true دارد ، این به این معناست که تابع مورد نظر زمانی که صدا زده می‌شود با چک کردن مقدار deriv بتواند متوجه شود که باید عملیات مربوط به back propagation را

انجام دهد یا forward propagation ، توضیحات بیشتر در مورد جزئیات اتفاقاتی که در زمان backpropagation در هر لایه می‌افتد در زمانی که در مورد توابع مورد نظر صحبت می‌کنیم، داده خواهد شد:

```
    def backward_prop(self, softmax_output, output_label):
        l8_delta          = (output_label - softmax_output) / softmax_output.shape[0]
        l7_delta          = self.relu(self.l8, l8_delta, deriv=True)              # (batch_sz, 10)
        l6_delta, self.fc2  = self.fully_connect(self.l6, self.fc2, l7_delta, deriv=True)  # (batch_s
z, 200)
        l5_delta          = self.relu(self.l6, l6_delta, deriv=True)              # (batch_sz, 200)
        l4_delta, self.fc1  = self.fully_connect(self.l4, self.fc1, l5_delta, deriv=True)  # (batch_s
z, 16, 4, 4)
        l3_delta          = self.mean_pool(self.l3, self.pool2, l4_delta, deriv=True)   # (batch_sz,
 16, 8, 8)
        l2_delta, self.conv2 = self.convolution(self.l2, self.conv2, l3_delta, deriv=True)  # (batch
_sz, 6, 12, 12)
        l1_delta          = self.mean_pool(self.l1, self.pool1, l2_delta, deriv=True)   # (batch_sz,
 6, 24, 24)
        l0_delta, self.conv1 = self.convolution(self.l0, self.conv1, l1_delta, deriv=True)  # (batch
_sz, 1, 28, 28)
```

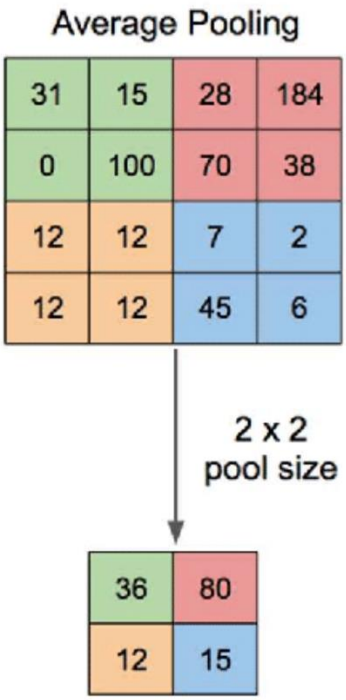
تابع convolution : یکی از مدل لایه‌هایی که در شبکه عصبی lenet استفاده می‌شود تابع Convolution است ، اولین متغیری که در این تابع استفاده شده start_time_convol است که برای اندازه گیری زمان اجرای این لایه از آن استفاده می‌کنیم که کاربرد اصلی آن در ادامه توضیح داده خواهد شد، متغیر N نشان دهنده تعداد شماره عکس ورودی، C نشان دهنده تعداد کانال هر عکس ورودی، W نشان دهنده عرض عکس ورودی و H نشان دهنده‌ی ارتفاع عکس ورودی است، K_NUM نشان دهنده شماره فیلتر ، K_C نشان دهنده تعداد کانال هر فیلتر، K_W نشان دهنده عرض فیلتر و K_Hنشان دهنده‌ی ارتفاع فیلتر است، اگر در فاز forward_propagation باشیم یعنی deriv == false باشد آنگاه ابتدا یک آرایه به اندازه ابعاد خروجی لایه Convolution با مقدار دهی اولیه ۰ می‌سازیم و اسم آن را feature_map می‌گذاریم ، حال برای انجام عمل

Convolution روی عکس ورودی به سه حلقه for احتیاج داریم برای طی کردن مراحل و در نهایت در هر مرحله از اجرای حلقه‌ها برای محاسبه Convolution از تابع convolve2d که جلو تر در مورد آن توضیح خواهیم داد استفاده می‌کنیم، نکته مهم این تابع بخش پیاده سازی CB است که کد آن چند خط کد زیر مجموعه if(w==28) است ، در اینجا تابع CB صدا زده می‌شود که مهم‌ترین بخش برنامه است و در ادامه به طور مفصل کاکرد این تابع را بررسی خواهیم کرد ولی به طور خلاصه این تابع اگر خطای در عکس لایه Convoltion رخ داده باشد تشخیص می‌دهد و به عکس برچسب U/ می‌زند ، اگر در فاز back bropagation باشیم و deriv == true باشد ابتدا یک ماتریس با مقادیر صفر و با ابعاد عکس اولیه ایجاد می‌کنیم، سپس یک ماتریس با مقادیر صفر و با ابعاد فیلتر ایجاد می‌کنیم، بعد ماتریس padded_frontdelta که از روی ماتریس pad شده front_delta که در مورد آن بعدتر صحبت خواهیم کرد ایجاد می‌شود، سپس ماتریس back_delta با ضرب کانولوشن padded_frontdelta و فیلتر ، با استفاده از ۳ حلقه for ایجاد می‌شود ، و ماتریس kernel_gradiant هم با ضرب کانولوشن front_delta و input_mapمقدار ده می‌شود و در نهایت فیلتر با مقادیر جدید kernel_gradient که در نرخ آموزش ضرب می‌شود، جمع می‌شود و بروز رسانی می‌شود:

```
def convolution(self, input_map, kernal, front_delta=None, deriv=False):
    global start_time_convol
    start_time_convol = time.time()
    N, C, W, H = input_map.shape#(batch,channel,w,h)
    K_NUM, K_C, K_W, K_H = kernal.shape
    if deriv == False:
        feature_map = np.zeros((N, K_NUM, W-K_W+1, H-K_H+1))
        for imgId in range(N):#we need image ID to build label array
            for kId in range(K_NUM):
                for cId in range(C):
                    feature_map[imgId][kId] += \
                        convolve2d(input_map[imgId][cId], kernal[kId,cId,:,:], mode='valid')
##########################################################################
#################################
        if (W == 28) :#if first convolution layer
```

```
                CB(imgId , d[num_iter_cnn] , Uth[num_iter_cnn] , input_map[imgId][cId] , featur
e_map[imgId][kId], kernal[kId,cId,:,:] )#we call CB here CB's inputs:(i,d,Uth,in_image,out_im
age,filter)

  ##########################################################################
##################################

            global end_time_convol

            end_time_convol = time.time()

            return feature_map

        else :

            # front->back (propagate loss)

            back_delta = np.zeros((N, C, W, H))

            kernal_gradient = np.zeros((K_NUM, K_C, K_W, K_H))

            padded_front_delta = \

              np.pad(front_delta, [(0,0), (0,0), (K_W-1, K_H-1), (K_W-1, K_H-
1)], mode='constant', constant_values=0)

            for imgId in range(N):

               for cId in range(C):

                  for kId in range(K_NUM):

                     back_delta[imgId][cId] += \

                       convolve2d(padded_front_delta[imgId][kId], kernal[kId,cId,::-1,::-
1], mode='valid')

                     kernal_gradient[kId][cId] += \

                       convolve2d(front_delta[imgId][kId], input_map[imgId,cId,::-1,::-
1], mode='valid')

            # update weights

            kernal += self.lr * kernal_gradient

            return back_delta, kernal
```

تابع mean_pool : یکی از لایه‌هایی که در شبکه lenet بارها از آن استفاده می‌شود pooling است که وظیفه کلی آن کاهش اندازه عکسی است که در مسیر شبکه عصبی در حال حرکت است تابع mean_pool همانطور که از اسمش پیداست کار کاهش اندازه ماتریس را با میانگین گرفتن از چند خانه کنار هم و نتیجه آن‌ها را به عنوان یک خانه خروجی در نظر گرفتن، انجام می‌دهد، این تابع خود از تابع block_reduce که از کتابخانه skimage.measure آورده شده ، استفاده می‌کند که در واقع همین وظیفه کاهش اندازه به وسیله میانگین گیری را انجام می‌دهد، شکل ۵ که قبل از کد تابع mean_pooling قرار گرفته به خوبی عملکرد mean_pooling را نمایش می‌دهد.



شکل ۵ : pooling با میانگین گیری

```
def mean_pool(self, input_map, pool, front_delta=None, deriv=False):
    N, C, W, H = input_map.shape
    P_W, P_H = tuple(pool)
    if deriv == False:
       feature_map = np.zeros((N, C, int(W/P_W), int(H/P_H)))
```

```
        feature_map = block_reduce(input_map, tuple((1, 1, P_W, P_H)), func=np.mean)
        return feature_map
    else :
        # front->back (propagate loss)
        back_delta = np.zeros((N, C, W, H))
        back_delta = front_delta.repeat(P_W, axis = 2).repeat(P_H, axis = 3)
        back_delta /= (P_W * P_H)
        return back_delta
```

تابع fully_connected : لایه‌های fc) fully_connected) که دو لایه انتهایی شبکه lenet را تشکیل می‌دهند به صورت کلی وظیفه کلاس بندی نهایی با تبدیل کلاس عکس به فرم one_hot را بر عهده دارند ، همانطور که در تابع‌های قبلی هم دیدیم عملکرد این تابع هم در فاز backward propagation با froward propagation متفاوت است در فاز forward propagation خروجی نهایی با ضرب نقطه ای [۳۵] ماتریس ورودی لایه که به صورت خطی در آمده در ماتریس fc که در ___init_ مقدار دهی شده بدست می‌آید و در فاز backpropagation ابتدا ماتریس front_delta در ترانهاده fc ضرب نقطه ای می‌شود و سپس ماتریس حاصل به ابعاد ماتریس ورودی اصلی لایه در می‌آید و در نهایت ماتریس fc که همان ماتریس وزن است با ضرب نقطه ای فرم خطی شده ماتریس داده ورودی در ماتریس front_delta بدست می‌آید :

```
def fully_connect(self, input_data, fc, front_delta=None, deriv=False):
    N = input_data.shape[0]
    if deriv == False:
        output_data = np.dot(input_data.reshape(N, -1), fc)
        return output_data
    else :
        # front->back (propagate loss)
        back_delta = np.dot(front_delta, fc.T).reshape(input_data.shape)
        # update weights
```

[35] Dot product

```
        fc += self.lr * np.dot(input_data.reshape(N, -1).T, front_delta)
        return back_delta, fc
```

تابع relu که جزو توابع فعالسازی[۳۶] است و روی خروجی توابع fc در شبکه lenet اعمال می‌شود ، اگر مقدار عدد x مثبت باشد خود عدد و اگر منفی باشد صفر را بر می‌گرداند و در فاز backpropagation مقدار عدد x مثبت باشد back_delta همان front_delta می‌شود :

```
def relu(self, x, front_delta=None, deriv=False):
    if deriv == False:
        return x * (x > 0)
    else :
        # propagate loss
        back_delta = front_delta * 1. * (x > 0)
        return back_delta
```

تابع softmax : Softmax یک تابع ریاضی است که بردار اعداد را به بردار احتمالات تبدیل می‌کند، جایی که احتمالات هر مقدار متناسب با مقیاس نسبی هر مقدار در بردار است. رایج‌ترین استفاده از تابع softmax در یادگیری ماشین کاربردی، استفاده از آن به عنوان یک تابع فعال سازی در مدل شبکه عصبی است. به طور خاص، شبکه برای خروجی N مقدار، یک عدد برای هر کلاس در کار طبقه‌بندی، پیکربندی شده است، و تابع softmax برای عادی سازی خروجی‌ها استفاده می‌شود، و آنها را از مقادیر مجموع وزن‌دار به احتمالات مجموع به یک تبدیل می‌کند. هر مقدار در خروجی تابع softmax به عنوان احتمال عضویت برای هر کلاس تفسیر می شود. تابع Softmax اغلب در شبکه‌های عصبی استفاده می‌شود تا نتایج لایه خروجی را که نرمال نشده است به یک توزیع احتمال بر روی کلاس‌های خروجی پیش‌بینی‌شده نگاشت کند. کد پیاده سازی این تابع در زیر آمده، ابتدا یک لیست به نام y ایجاد می‌کنیم ، سپس با یک حلقه روی داده‌های ورودی به نام x حرکت می‌کند و در هر گام در حلقه e به توان ،)t که یک ردیف از x است) منهای بیشینه مقدار در t می‌کند و سپس احتمال معادل آن را در خانه‌ی متناظر در لیست y اضافه می‌کند :

```
def softmax(self, x):
    y = list()
```

[36] Activation

```
    for t in x:
        e_t = np.exp(t - np.max(t))
        y.append(e_t / e_t.sum())
    return np.array(y)
```

حال به معرفی توابع باقیمانده می‌پردازیم و بعد از آن برنامه را برای آموزش اجرا می‌کنیم و خروجی‌ها را بررسی می‌کنیم.

معرفی تابع xavier_init : Xavier تلاشی برای بهبود مقداردهی اولیه ورودی‌های وزنی شبکه عصبی، به منظور جلوگیری از برخی مشکلات سنتی در یادگیری ماشین است. در اینجا، وزن‌های شبکه برای مقادیر میانی خاصی انتخاب می‌شوند که در کاربرد یادگیری ماشین مفید هستند. وزن‌ها در مقدار دهی xavier [۳۷] با توزیع $\left(-\sqrt{\frac{6}{n_i+n_o}}, \sqrt{\frac{6}{n_i+n_o}}\right)$ محاسبه می‌شوند. تابع زیر یک ماتریس با ابعاد c1,c2,w,h درست می‌کند که مقادیر درایه‌های آن بین ۰ و یک است سپس همه‌ی خانه‌های ماتریس را منهای ۱ می‌کند و در ۲ ضرب می‌کند تا اعداد بین -۱ و ۱ شوند و در نهایت در ratio که همان رادیکال تقسیم ۶ بر fan_1 (تعداد ورودی لایه) بعلاوه fan_2 تعداد خروجی از لایه) می‌کند.

پیاده سازی تابع ساده xavier به صورت زیر است :

```
def xavier_init(c1, c2, w=1, h=1, fc=False):
    fan_1 = c2 * w * h
    fan_2 = c1 * w * h
    ratio = np.sqrt(6.0 / (fan_1 + fan_2))
    params = ratio * (2*np.random.random((c1, c2, w, h)) - 1)
    if fc == True:
        params = params.reshape(c1, c2)
    return params
```

[37] Xavier initialization

معرفی تابع shuffle_dataset : این تابع ابتدا تعداد کل تصاویر (N) را می‌گیرد و سپس به صورت رندوم یک جایگشت از اعداد ۰ تا N را ایجاد می‌کند ، سپس ماتریس جدید x را برابر ماتریس داده‌های ورودی که خانه‌های آن به صورت جایگشتی از N جابجا شده ، قرار می‌دهد و برای ماتریس برچسب عکس‌ها هم چنین می‌کند. کد این تابع بع صورت زیر است:

```
def shuffle_dataset(data, label):
    N = data.shape[0]
    index = np.random.permutation(N)
    x = data[index, :, :]; y = label[index, :]
    return x, y
```

حال به معرفی اصلی‌ترین تابع در پیاده سازی مدل معرفی شده در مقاله می‌پردازیم:

تابع CB (بلوک کنترلی) : این تابع که از بخش‌های مختلفی تشکیل شده اصلی‌ترین بخش برنامه را تشکیل می‌دهد، ورودی‌های این تابع همان ورودی هاییست که در شکل ۳ دیدیم (in_img ، out_img) به استثنای اینکه ما در اینجا img_id یعنی شماره عکسی که به این تابع می‌دهیم، d که همان پارامتر میزان کوچک شدن است، Uth که حد آستانه U است و فیلتر اصلی را به نام filter ، یه این تابع پاس می‌دهیم ; ابتدا در این تابع عکسی که هنوز خطا نخورده چاپ می‌شود تا بعد با عکس خطا خورده آن را مقایسه کنیم، سپس دو عکس با ابعاد کوچک‌تر تولید می‌شود که فیلتر تقریبی مدل ما را اجرا کنند، میزان کوچک شدن عکس‌ها به پارامتر d که به این تابع پاس دادیم بستگی دارد، برای کوچک کردن تصاویر از همان تکنیک که درمورد تابع mean_pool توضیح دادیم استفاده می‌کنیم و تابع block_reduce را به کار می‌بریم، پس از کوچک کردن عکس‌ها با استفاده از عملیات convolution فیلترهای تقریبی را روی این دو عکس که دقیقاً کپی یکدیگر هستند اعمال می‌کنیم، در فاز بعدی باید دو عکس خروجی حاصل از عملیات convolution را با هم مقایسه کنیم، اگر این دو با هم دقیقاً برابر بودند به سراغ مرحله بعدی که برچسب گذاری عکس‌ها به صورت U یا /U است می‌رویم،نکته مهم در اینجا این است که چون ما غعلا در فاز آموزش هستیم و می‌خواهیم مقادیر و ترکیب‌های مختلف Uth و d را امتحان کنیم و بهترین آن‌ها بر اساس کمترین میزان D/U در کلاس بندی نهایی ، را انتخاب کنیم ما به دو عکسی که فیلتر تقریبی روی آن‌ها خورده خطایی تزریق نمی‌کنیم تا حتماً وارد فاز بعدی بشویم.

حال که خطایی تزریق نشده، حاصل اعمال دو فیلتر تقریبی یکسان است در این مرحله فرض می‌کنیم که به out_img که حاصل خروجی فیلتر اصلی است خطا تزریق شده بنابراین بعضی از بیت‌های آن را به صورت رندوم تغییر می‌دهیم، دلیل اینکه فرض می‌کنیم خطا تزریق شده هم این است که ما می‌خواهیم مقادیر و ترکیب‌های

مختلف Uth و d را امتحان کنیم و بهترین آن‌ها بر اساس کمترین میزان D/U/ در کلاس بندی نهایی ، را انتخاب کنیم بنابراین باید سعی کنیم عکس‌هایی که غیر قابل استفاده هستند تولید کنیم، در گام بعدی عکس‌هایی که خطا خورده اند را چاپ می‌کنیم تا در نهایت کاربر نهایی (در اینجا نویسنده گزارش کاربر نهاییست ولی در مقاله از یک ابزار به نام oracle استفاده کرده) بتواند تشخیص دهد که آیا واقعاً کلاس بندی U/ وU درست بوده ؟ و در نهایت نتایج را در ۴ دسته D/U/ ، DU/ ، DU و D/U قرار دهد ، حال نوبت برچسب گذاری عکس‌ها توسط sc است ، در این مرحله بیت به بیت عکس خروجی فیلتر اصلی (خطا خورده) و عکس تقریبی خطا خورده با هم مقایسه می‌شوند، اگر اختلاف دو بیت متناظر از هم بیشتر از ۰.۰۵ (نرخی که خود مقاله معرفی کرده) باشد به شمارنده counter_faulty_pixe یک واحد اضافه می‌شود، پس از انجام عمل مقایسه اگر نسبت تعداد پیکسل‌های خطا خورده (counter_faulty_pixels) به تعداد کل پیکسل‌های یکی از دو عکس (ابعاد دو عکس برابر است) بیشتر از 1-Uth باشد به خروجی sc برچسب U/ است و لایه متناظر باید مجداد اجرا گردد در غیر اینصورت نتیجه نهایی سالم است و sc برچسب U را بر می‌گرداند ، ما این برچسب‌ها را برای عکس‌های مختلف درون آرایه labels نگه داشته ایم تا در انتها آن را چاپ کنیم و بتوانیم ارزیابی روی نتیجه نهایی را انجام دهیم:

```
def CB(img_id,d,Uth,in_image,out_image,filter):


    # Plotting output images of convolution layer to compare with faulty output
    plt.figure(figsize=(5,5))
    plt.subplot(321)
    plt.imshow(out_image.squeeze(),cmap='gray')
    print('main filter output# :',img_id)
    plt.axis('off')
    plt.show()


    #resizing image 2 times apx_img1 apx_img2
    height = len(in_image)
    width = len(in_image[0])
    apx_img_height =int(height / d)
```

```
    apx_img_width = int(width / d)
    apx_img1 = np.zeros((int(width/d), int(height/d)))
    apx_img1 = block_reduce(in_image, tuple(( d, d)), func=np.mean)
    apx_img2 = np.zeros((int(width/d), int(height/d)))
    apx_img2 = block_reduce(in_image, tuple(( d, d)), func=np.mean)

    #implementing same conv2D on these apx_img1 apx_img2
    K_W, K_H = filter.shape

    apx_img1_convlvd = np.zeros((apx_img_width-K_W+1, apx_img_height-
K_H+1)) #10 * 10
    apx_img2_convlvd = np.zeros((apx_img_width-K_W+1, apx_img_height-
K_H+1)) #10 * 10

    apx_img1_convlvd += \
      convolve2d(apx_img1, filter[:,:], mode='valid')
    apx_img2_convlvd += \
      convolve2d(apx_img2, filter[:,:], mode='valid')

    #check every pixel of apx_img1 and apx_img2 equal
    is_euqal = np.array_equiv(apx_img1, apx_img2)

    #if equal = false recompute Conv2D layer (/D)

    #if equal = true go to next line
```

```
if (is_euqal):
  out_img_height = len(out_image)
  out_img_width = len(out_image[0])
  #change random number =(num_chnge_bit)  pixel(s) of out_image
  num_chnge_bit = random.randint(out_img_height * out_img_width)
  faulty_out_image = np.copy(out_image)
  for num_fault in range(num_chnge_bit):
     #generate two random number to inject fault in out_image
     i = random.randint(out_img_height-1)
     j = random.randint(out_img_width-1)
     rand_val = random.uniform(-1, 1)
     faulty_out_image[i,j] = rand_val

  # Plotting fault injected images of convolution layer to compare with faulty output
  plt.figure(figsize=(5,5))
  plt.subplot(321)
  plt.imshow(faulty_out_image.squeeze(),cmap='gray')
  print('faulty output# :',img_id)
  plt.axis('off')
  plt.show()
  counter_faulty_pixels = 0
  for i in range(out_img_height):
    for j in range(out_img_width):
      if (abs((faulty_out_image[i,j] - out_image[i,j])) > 0.05):#the Manhattan distance of the el
ement of output and faulty output is lower than a set threshold 0.05.
```

```
            counter_faulty_pixels += 1
        #define label of image
        print("counter_faulty_pixels / (out_img_height * out_img_width)",counter_faulty_pixels / (o
ut_img_height * out_img_width))
        if(counter_faulty_pixels / (out_img_height * out_img_width) > 1-Uth):
          label[num_iter_cnn][img_id] = '/U'
        else:
          label[num_iter_cnn][img_id] = 'U'
        return
```

۴.۷.مرحله ارزیابی و انتخاب Uth و d مناسب :

در این مرحله ما برنامه را روی حالت‌های مختلف ترکیب Uth وd های مختلف (جمعاً ۴ حالت) اجرا می‌کنیم، سپس برای هر یک از ۴ مرجله خروجی آرایه labels را مشاهده می‌کنیم و با مقایسه برچسب‌های /U وU که در این آرایه قرار گرفته و عکس‌هایی که چاپ شده اند نتایج را در ۴ گروه /D/U ، /DU ، DU و D/U دسته بندی می‌کنیم ، و در نهایت ترکیبی از Uth و d که کمترین میزان /D/U را داشته باشد برای مدل اصلی انتخاب می‌شود، توجه داشته باشید ما در اینجا cb را برای ۱ لایه قرار دادیم تا از پیچیدگی زیاد و بی مورد پرهیز کنیم اگر بنا باشد برای همه لایه‌ها cb را قرار دهیم مراحل گفته شده دقیقاً مانند قبل طی می‌شود ولی تنها دو تفاوت وجود دارد، ۱ – تابع cb در تمام لایه‌هایی که می‌خواهیم برای آن cb قرار دهیم صداد زده می‌شود، ۲ – در نتیجه گیری نهایی به ازای هر ترکیبی از Uth وd های مختلف که در نظر می‌گیریم، برای هر لایه یک آرایه labels تولید می‌شود که ما از روی آن لایه و عکس‌های خروجی چاپ شده می‌توانیم تعداد /D/U را بدست بیاوریم ، حالا بین تعداد /D/U در تمام لایه‌ها میانگین می‌گیریم و انتخاب ترکیب نهایی Uth و d بر اساس این میانگین‌ها است.

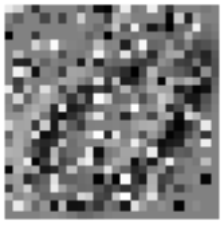
ما در مرحله ارزیابی دو مقدار ۰.۴۵ و ۰.۸۵ را برای Uth و دو مقدار ۲ و ۴ را برای d در نظر گرفتیم که ترکیب این ها با هم ۴ حالت را می‌سازد که می‌توانید در کد زیر آرایه‌هایی که تعریف شده اند را ببینید:

```
d = [2 , 4 , 4 , 2]
Uth = [0.85 , 0.45 , 0.85 , 0.45]
```

در مرحله i از ارزیابی خانه i ام آرایه d به عنوان مقدار d و خانه i ام آرایه Uth به عنوان Uth انتخاب می‌شوند و به تابع cb پاس داده می‌شوند ، برای کاهش پیچیدگی و سهولت بررسی ما الگوریتم را روی ۱۰ داده آموزششی از پایگاه داده mnist اجرا کرده ایم ، بخشی از خروجی برنامه به صورت زیر است (برای پرهیز از حشو و طولانی شدن گزارش تمام خروجی را در این گزارش نیاورده ایم و خروجی برنامه را ضمیمه فایل ارسالی کرده ایم):

main filter output# : 0

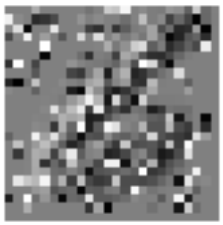
faulty output# : 0



counter_faulty_pixels / (out_img_height * out_img_width) 0.4149305555555556

main filter output# : 1

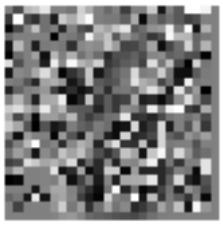
faulty output# : 1



counter_faulty_pixels / (out_img_height * out_img_width) 0.3298611111111111

main filter output# : 2

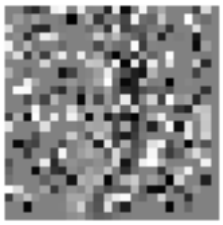
faulty output# : 2



counter_faulty_pixels / (out_img_height * out_img_width) 0.5746527777777778

main filter output# : 3

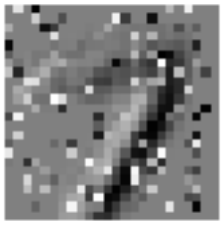
faulty output# : 3



counter_faulty_pixels / (out_img_height * out_img_width) 0.4392361111111111

main filter output# : 4

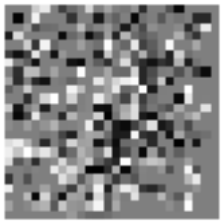
faulty output# : 4



counter_faulty_pixels / (out_img_height * out_img_width) 0.2065972222222222

main filter output# : 5

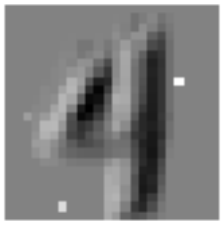
faulty output# : 5



counter_faulty_pixels / (out_img_height * out_img_width) 0.46875

main filter output# : 6

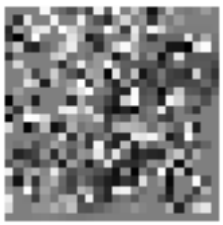
faulty output# : 6



counter_faulty_pixels / (out_img_height * out_img_width) 0.005208333333333333

main filter output# : 7

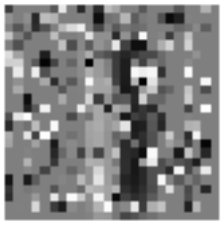
faulty output# : 7



counter_faulty_pixels / (out_img_height * out_img_width) 0.5173611111111112

main filter output# : 8

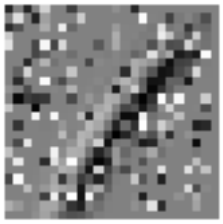
faulty output# : 8



counter_faulty_pixels / (out_img_height * out_img_width) 0.390625

main filter output# : 9

faulty output# : 9



counter_faulty_pixels / (out_img_height * out_img_width) 0.2847222222222222

label[num_iter_cnn][] : ['/U', '/U', '/U', '/U', '/U', '/U', 'U', '/U', '/U', '/U']

total execution time of double execution of CNN : 1.8011231422424316 (seconds)

start_time_convol : 1675578081.321698

end_time_convol : 1675578081.2724423

execution time of program with approximate base technique : 0.8513059616088867 (seconds)

همانطور که می‌بینید فرضاً برای حالت اول که Uth = 0.85 و d = 2 است آرایه labels به صورت زیر می‌باشد:

label[1][:] : ['/U', '/U', '/U', '/U', '/U', '/U', 'U', '/U', '/U', '/U']

با بررسی عکس‌های چاپ شده می‌بینیم مقدار D/U/ برای این حالت برابر صفر است، برای سایر حالات نتیجه به صورت زیر است:

حالت دوم (Uth = 0.45 و d = 4) : مقدار D/U/ برابر ۵ است.

حالت سوم (Uth = 0.85 و d = 4) : مقدار D/U/ برابر 0 است.

حالت چهارم (Uth = 0.45 و d = 2) : مقدار D/U/ برابر 8 است.

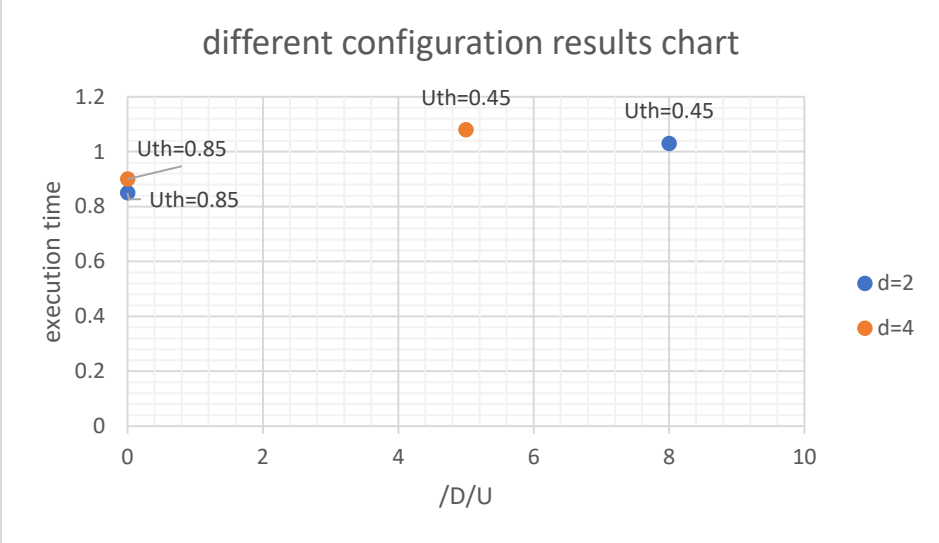
از نتایجی که بدست آمده مشاهده می‌کنیم حالت سوم و اول کمترین D/U/ را دارند و با توجه به زمانی که این دو اجرا شده اند (زمان اجرا معیار دوم مقاله برای انتخاب از بین حالات است)، زمان اجرای حالت سوم برابر 0.9087789058685303 و زمان اجرای حالت اول برابر 0.8513059616088867 دارد ، که در این اجرا حالت اول عملکرد بهتری داشته.

۵)نتیجه‌گیری:

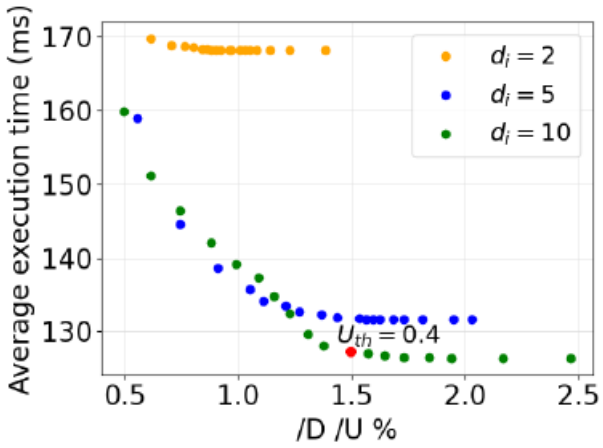
از بررسی نتایج به‌دست‌آمده از اجرای مدل معرفی شده می‌توانیم ببینیم اگر می‌خواستیم از تکنیک DWC استفاده کنیم و کل شبکه عصبی را در هر لایه دو مرتبه اجرا کنیم تا خطا را متوجه شویم زمان اجرای برنامه ۲ برابر می‌شد، اما در اینجا با توجه به اینکه در هر لایه مقایسه روی داده‌های تقریبی انجام می‌شود (با اندازه کوچک‌تر از اندازه داده اصلی) حجم محاسبات برای تشخیص خطا تا حد زیادی کاهش می‌یابد و همچنین فقط به ازای داده‌هایی که U/ هستند مجبوریم فقط آن لایه ای که خطا خورده را دوباره اجرا کنیم در نتیجه به طور منطقی قطعاً زمان اجرا تا حد بسیار زیادی پایین می‌آید، همانطور که در پیاده سازی این تکنیک هم می‌بینیم زمان اجرا برای پیاده سازی که ما انجام دادیم نسبت به روش DWC تقریباً نصف است ، که این نشان دهنده

کارایی بالای مدل پیشنهادی مقاله است، همچنین دقت هم در این مدل بسیار بالاست، چرا که در نتایج بدست آمده در پیاده سازی هم دیدیم که مقدار D/U/ منفی‌های اشتباه صفر بوده و خطایی که خود مقاله اعلام کرده که در پیاده سازی در ابعاد بالاتر به آن رسیده تنها ۴ درصد است که این نکته مثیت دیگر این مدل است.

در نتایج به‌دست‌آمده در ۴ حالت پیاده‌سازی شده توسط نویسنده گزارش همان‌طور که انتظار می‌رفت با افزایش Uth دقت افزایش می‌یابد (به این معنی که تعداد D/U/ ها کاهش می‌یابد) و با کاهش d زمان اجرا افزایش می‌یابد ، در این اجرا چون تعداد خطاهای تزریق شده به‌صورت رندوم هستند علت اینکه زمان اجرا d = 4 از d = 2 بیشتر شده این است که تعداد خطاهایی که تزریق شده در d = 4 بیشتر بوده و حلقه for مربوط به تزریق خطا بیشتر اجرا شده، شکل ۶ نمودار ۴ حالت به‌دست‌آمده برای Uth و dها را نشان می‌دهد که محور x آن نشان‌دهنده مقدار D/U/ و محور y آن نشان‌دهنده زمان اجراست ، همان‌طور که می‌بینید نتیجه به‌دست‌آمده با شکل ۷ که از مقاله اصلی گرفته شده تطابق منطقی ای دارد.



شکل ۶ : نمودار نتیج بدست آمده برای نمودار ۴ حالت به‌دست‌آمده برای Uth و dهای متفاوت



شکل ۷ : نمونه ای از پیکربندی های مختلف ارائه شده توسط مقاله.

۶)فهرست مراجع

[1] J. Andersson, M. Hjorth, F. Johansson, and S. Habinc, “LEON Processor Devices for Space Missions: First 20 Years of LEON in Space,” in Int. Conf. Space Mission Challenges for Inform. Tech., 2017,pp. 136–141

[2] M. Fayyaz and T. Vladimirova, “Fault-tolerant distributed approach to satellite on-board computer design,” in Aerospace Conf.,2014, pp. 1–12.

[3] M. Biasielli, C. Bolchini, L. Cassano, E. Koyuncu, and A. Miele, “A Neural Network Based Fault Management Scheme for Reliable

Image Processing,” IEEE Trans. Computers, vol. 69, no. 5, pp. 764– 776, 2020.

پایان