به نام خدا

گزارش پروژه شبیه سازی

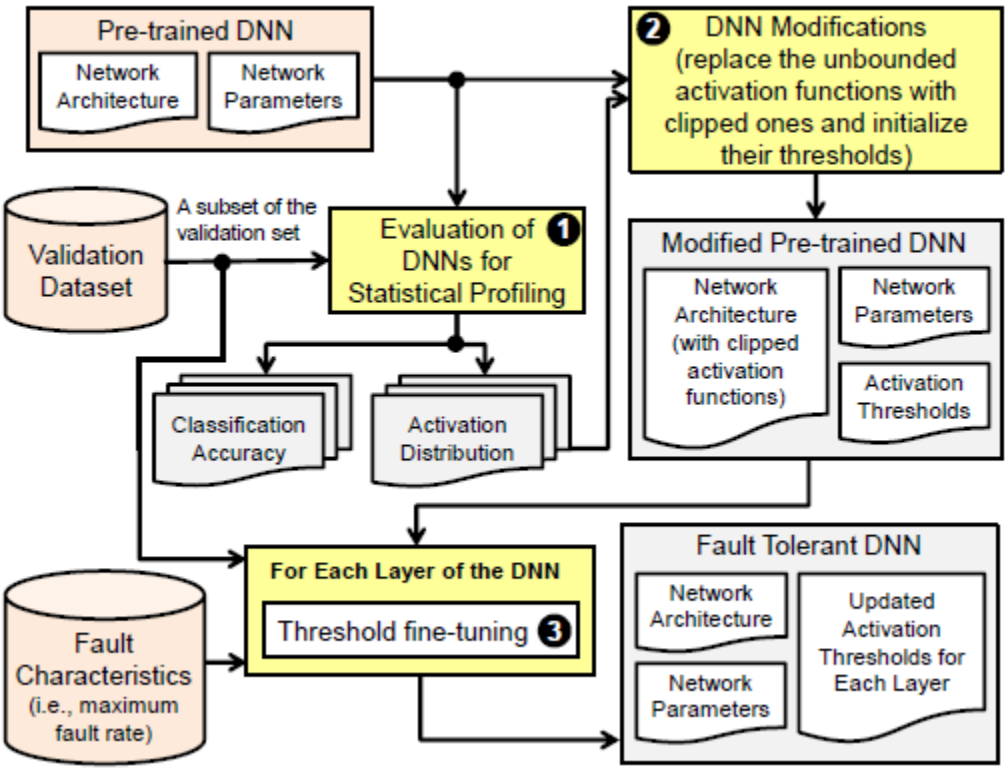
دیبا مسیحی ۹۷۱۱۰۲۷٤

مهسا امانی ۹۷۱۰۵۷٦۹

گروه ۱۰

در این پروژه مقاله زیر شبیه سازی شد:

FT-ClipAct: Resilience Analysis of Deep Neural Networks and Improving their Fault Tolerance using Clipped Activation



روند کلی پروژه بصورت زیر است:

۱- لود کردن دیتاست

دیتاست مورد استفاده Cifar ۱۰ نام دارد که ۵۰۰۰۰ داده آموزشی و ۱۰۰۰۰ داده تست دارد. این دیتاست بصورت آماده در کتابخانه pytorch وجود داشته و قابل استفاده است. در مقاله گفته شده که تنها بخشی از داده های validation استفاده شده است، پس از داده های آموزش ۱۰۰۰ تای آن ها را به این منظور انتخاب می کنیم.

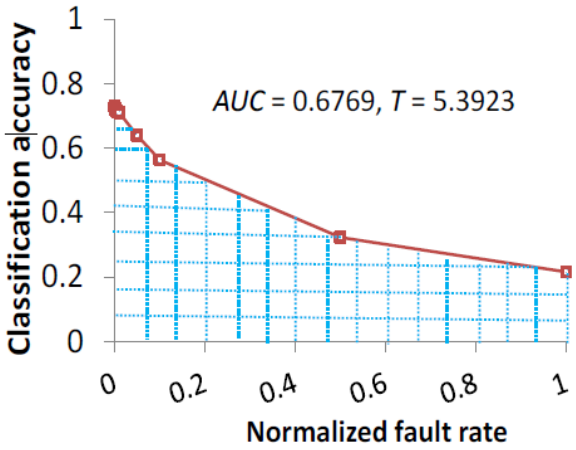
۲- تعریف **Activation function** جدید

برای اینکه اثر خطاهایی که در ذخیره وزن های شبکه رخ می دهند به حداقل برسد، activation function جدیدی تعریف می کنیم که بصورت زیر است:

$$f(x) = \begin{cases} x, & \text{if } 0 \le x \le T \\ 0, & \text{otherwise} \end{cases}$$

که بهترین مقدار T را در مراحل بعد بدست خواهیم آورد.

۳- تعریف معیار ارزیابی



حال برای اینکه بهترین T ممکن را انتخاب کنیم نیاز داریم تا معیاری برای ارزیابی داشته باشیم. برای نرخ خطاهای مختلف شبکه دقت های متفاوتی به ما می دهد، پس هنگامی که T های متفاوتی با استفاده از الگوریتم fine tuning بررسی شد، برای هر T و با تست نرخ خطاهای مختلف دقت هایی را بدست خواهیم آورد که در صورت رسم نمودار دقت بر حسب نرخ خطا، مساحت زیر نمودار که با روش ذوزنقه ای حساب می شود و آن را AUC می نامیم، برای آن T بدست می آید.

٤- استفاده از فریم ورک **Ares**

کدهای مربوط به این قسمت برای انجام تحلیل‌های مربوط به تزریق خطا(fault injection) در ares زده شده‌اند.

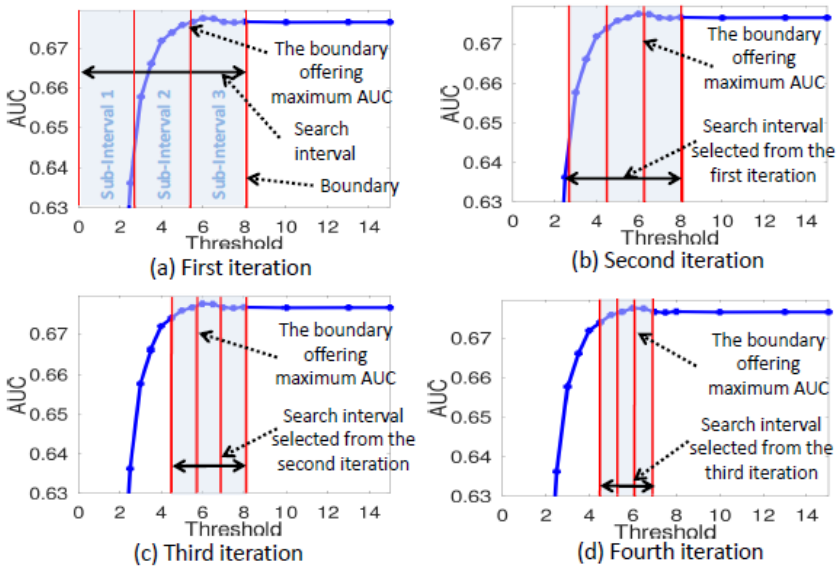
در واقع تزریق خطا را برای مدل‌های مختلف در وزن‌ها نشان میدهیم، این خطاهایی که تزریق میکنیم تصادفی هستند و برای این منظور دو تابع اصلی Inject و تابع fault wrapper را پیدا سازی کردیم. برای اطلاعات بیشتر به این مقاله می‌توانید مراجعه کنید.

٥- تعریف الگوریتم های **fine tuning**

برای fine tune کردن و یافتن بهترین مقدار T برای هر لایه از شبکه مان، از شبه کد های موجود در مقاله استفاده کردیم. در کل قابل توجه است که Activation function های هر لایه مقدار T متفاوتی خواهند داشت. روش الگوریتم به این صورت است که برای هر لایه از شبکه روند زیر را انجام می گیرد:

- شبکه های استفاده شده در این مقاله وزن های ترین شده دارند که در مرحله اول بزرگترین مقدار را تحت عنوان ACT_{max} و برای اولین T در نظر میگیریم.
- بازه ی بین ۰ تا ACTmax را به ۳ زیر بازه یکسان تقسیم می کنیم که ٤ تا T جدید بدست خواهد آمد(ابتدای هر بازه و انتهای آخرین بازه).
- هر کدام از T ها را در activation function مان تست می کنیم و دقت های آن ها را ذخیره می کنیم. با استفاده از نرخ خطاهای مختلف (چون در مقاله دقیق ذکر نشده ما توان های -٥ تا -٨ از ۱۰ را در نظر گرفتیم.) و دقت های بدست آمده AUC را حساب می کنیم. پس تا اینجا بازای هر T یک AUC بدست آمد.
- آن بازه ای که T آن AUC ماکزیمم دارد، گزینه بهتری از مقدار قبلی آن برای activation function است و بازه مان را آپدیت می کنیم.

برای بازه جدید دوباره به مرحله ۲ می رویم و دوباره مراحل را تکرار می کنیم. تعداد دفعات تکرار را با یک حداقل، حداکثر و یا تفاوت میان ۲ AUC متوالی مشخص می کنیم. هر چقدر الگوریتم بیشتر تکرار شود یعنی بهتر fine tune می شود و نهایتا نتیجه بهتری بدست می دهد.



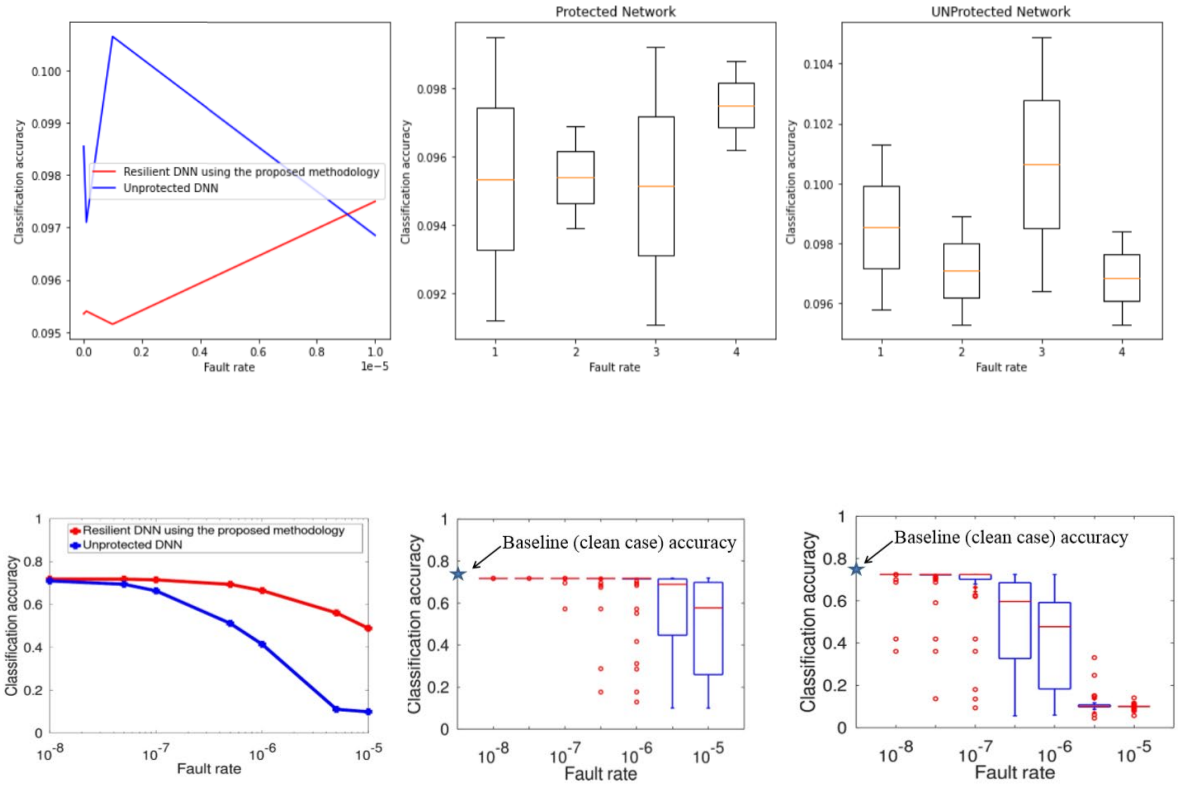
٦- معماری شبکه **AlexNet**

معماری شبکه در نوت بوک آپلود شده قابل مشاهده است و فقط توابع activation آن را با تابع custom ای که خودمان تعریف کردیم جایگزین می کنیم. چون شبکه برای تشخیص ۱۰۰۰ کلاس از هم است، پس از لود کردن وزن هایش لایه آخر را تغییر می دهیم و تعداد نورون های آن را به ۱۰ تغییر می دهیم.

۷- ارزیابی نتایج **AlexNet** و مقایسه با مقاله

پس از اجرای هفت لایه مدل alexnet و بدست آوردن داده‌های مورد نظر، این مدل را با thresholdهای جدید با مدل unprotected مقایسه کردیم و نتایج زیر را بدست آوردیم.

(توجه داشته باشید که در مقاله نتایج بدست آمده حاصل ۵۰ بار اجرای مدل بود، در حالی که ما به علت کمبود زمان برای ۵۰ بار اجرا و سنگین بودن مدل با هماهنگی با دستیار آموزشی درس این مدل را ۲ بار اجرا کردیم. الگوریتم fine tune کردن نیز یکبار فقط اجرا شد که در صورت اجرای بیشتر نتایج باز هم می توانست بهتر شود. تغییر تعداد نورون های لایه آخر نیز باعث شد که دقت پایین بیاید و نیاز بود تا دوباره train شود ولی به علت بزرگ بودن شبکه ممکن نبود و این مورد نیز مزید بر علت شد).



۸- معماری شبکه VGG۱٦

معماری شبکه در نوت بوک آپلود شده قابل مشاهده است و فقط توابع activation آن را با تابع custom ای که خودمان تعریف کردیم جایگزین می کنیم. چون شبکه برای تشخیص ۱۰۰۰ کلاس از هم است، پس از لود کردن وزن هایش لایه آخر را تغییر می دهیم و تعداد نورون های آن را به ۱۰ تغییر می دهیم.

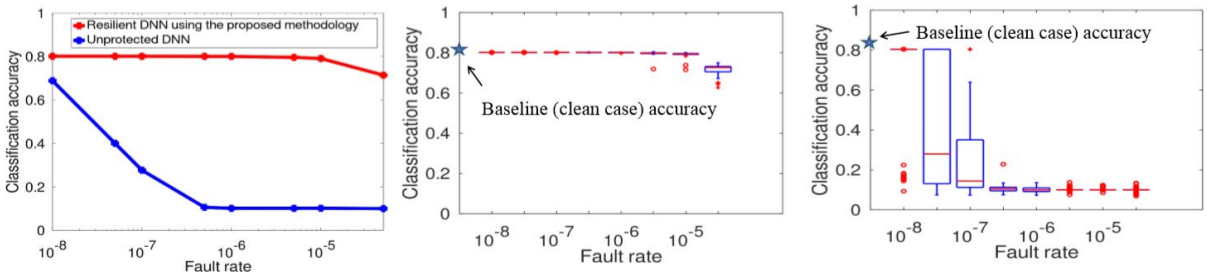
این معماری شامل ۱٦ لایه وزن دار است که ۱۳ لایه Convolutional است. این، یک شبکه بزرگ با تعداد پارامترهای بسیاری است. (مشکل مقاله: در مقاله گفته شده این شبکه ۱ لایه FC دارد در حالی که ۳ تا می باشد.)

۹- ارزیابی نتایج VGG۱٦ و مقایسه با مقاله

از نظر نحوه‌ی اجرا این مدل را مانند مدل alexnet اجرای آن را شروع کردیم اما به دلیل سنگین بودن زیاد این مدل و تعداد لایه‌های بسیار زیاد آن تنها توانستیم نهایتا پس از ۱۵ ساعت اجرا ۸ لایه از آن را اجرا کرده باشیم در حالی که با همان سرعت حداقل حدود ۲۵ ساعت اجرای آن طول می‌کشید که با هماهنگی دستیار آموزشی درس در این مقاله به رسیدن نتایج و اجرای مدل قبلی

اکتفا کردیم، هر چند که کدهای مربوط به این مدل و اجرای ۸ لایه‌ی آن را در این فایل می‌توانید مشاهده کنید.

نتایج مورد نظر مقاله برای این مدل به صورت زیر است.



۱۰- آیا مقاله جای بهبود دارد؟

با توجه به مشکلی که در بخش ۸ به آن اشاره شد، در صورتی که مقاله اشتباه کرده باشد، با دو لایه بیشتر FC می توان دقت های بهتری بدست آورد و احتمالا جای کار داشته باشد.