به نام خدا

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

درس شبکه‌های عصبی و یادگیری عمیق

تمرین چهارم

نام عضو اول	فاطمه رشیدی شهری
شماره دانشجویی	۶۱۰۳۹۹۱۳۱
نام عضو دوم	آراد وزیرپناه
شماره دانشجویی	۶۱۰۳۹۹۱۸۲

فهرست

شکل‌ها

جدول‌ها

قوانین

قبل از پاسخ دادن به پرسش‌ها، موارد زیر را با دقت مطالعه نمایید:

- از پاسخ‌های خود یک گزارش در قالبی که در صفحه‌ی درس در سامانه‌ی Elearn با نام ***REPORTS_TEMPLATE.docx*** قرار داده شده تهیه نمایید.
- پیشنهاد می‌شود تمرین‌ها را در قالب گروه‌های دو نفره انجام دهید. (بیش از دو نفر مجاز نیست و تحویل تک نفره نیز نمره‌ی اضافی ندارد) توجه نمایید الزامی در یکسان ماندن اعضای گروه تا انتهای ترم وجود ندارد. (یعنی، می‌توانید تمرین اول را با شخص A و تمرین دوم را با شخص B و ... انجام دهید)
- **کیفیت گزارش شما در فرآیند تصحیح از اهمیت ویژه‌ای برخوردار است**؛ بنابراین، لطفا تمامی نکات و فرض‌هایی را که در پیاده‌سازی‌ها و محاسبات خود در نظر می‌گیرید در گزارش ذکر کنید.
- در گزارش خود مطابق با آنچه در قالب نمونه قرار داده شده، برای شکل‌ها زیرنویس و برای جدول‌ها بالانویس در نظر بگیرید.
- الزامی به ارائه توضیح جزئیات کد در گزارش نیست، اما باید نتایج بدست آمده از آن را گزارش و تحلیل کنید.
- **تحلیل نتایج الزامی می‌باشد، حتی اگر در صورت پرسش اشاره‌ای به آن نشده باشد.**
- **دستیاران آموزشی ملزم به اجرا کردن کدهای شما نیستند**؛ بنابراین، هرگونه نتیجه و یا تحلیلی که در صورت پرسش از شما خواسته شده را به طور واضح و کامل در گزارش بیاورید. در صورت عدم رعایت این مورد، بدیهی است که از نمره تمرین کسر می‌شود.
- **کدها حتما باید در قالب نوت‌بوک با پسوند .ipynb تهیه شوند، در پایان کار، تمامی کد اجرا شود و خروجی هر سلول حتما در این فایل ارسالی شما ذخیره شده باشد.** بنابراین برای مثال اگر خروجی سلولی یک نمودار است که در گزارش آورده‌اید، این نمودار باید هم در گزارش هم در نوت‌بوک کدها وجود داشته باشد.
- **در صورت مشاهده‌ی تقلب امتیاز تمامی افراد شرکت‌کننده در آن، 100- لحاظ می‌شود.**
- تنها زبان برنامه نویسی مجاز **Python** است.
- **استفاده از کدهای آماده برای تمرین‌ها به هیچ وجه مجاز نیست. در صورتی که دو گروه از یک منبع مشترک استفاده کنند و کدهای مشابه تحویل دهند، تقلب محسوب می‌شود.**

- نحوه محاسبه تاخیر به این شکل است: پس از پایان رسیدن مهلت ارسال گزارش، حداکثر تا یک هفته امکان ارسال با تاخیر وجود دارد، پس از این یک هفته نمره آن تکلیف برای شما صفر خواهد شد.
  - سه روز اول: بدون جریمه
  - روز چهارم: ۵ درصد
  - روز پنجم: ۱۰ درصد
  - روز ششم: ۱۵ درصد
  - روز هفتم: ۲۰ درصد
- حداکثر نمره‌ای که برای هر سوال می‌توان اخد کرد ۱۰۰ بوده و اگر مجموع بارم یک **سوال** بیشتر از ۱۰۰ باشد، در صورت اخد نمره بیشتر از ۱۰۰، اعمال نخواهد شد.
  - برای مثال: اگر نمره اخذ شده از سوال ۱ برابر ۱۰۵ و نمره سوال ۲ برابر ۹۵ باشد، نمره نهایی تمرین ۹۷.۵ خواهد بود و نه ۱۰۰.
- لطفا گزارش، کدها و سایر ضمایم را به در یک پوشه با نام زیر قرار داده و آن را فشرده سازید، سپس در سامانه‌ی Elearn بارگذاری نمایید:

HW[Number] _[Lastname]_[StudentNumber]_[Lastname]_[StudentNumber].zip

(مثال: HW1_Ahmadi_810199101_Bagheri_810199102.zip)

- برای گروه‌های دو نفره، بارگذاری تمرین از جانب یکی از اعضا کافی است ولی پیشنهاد می‌شود هر دو نفر بارگذاری نمایند.

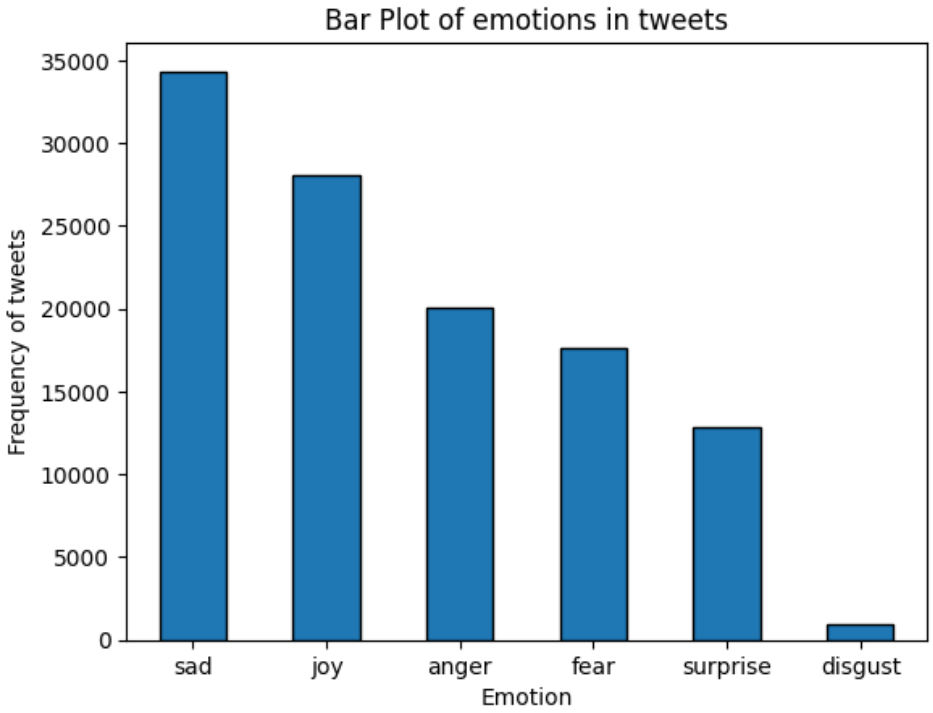
پرسش ۱. تحلیل احساسات متن فارسی

۱-۱. مجموعه داده

در این مجموعه از داده‌ها ۶ فایل csv موجود است که هر کدام توئیت‌های مربوط به ۶ احساس را دارند. احساس‌هایی که به دنبال طبقه بندی این دیتاست بر روی آن‌ها هستیم، به این صورت هستند:

- Sad
- Joy
- Anger
- Fear
- Surprise
- Disgust

از طرفی، می‌توانیم تعداد هر کدام از توئیت‌های مربوط به هر برچسب را در این دیتاست در نمودار میله‌ای زیر مشاهده کنیم.



شکل 1. نمودار میله‌ای مربوط به توزیع دیتاست

۱-۲. پیش‌پردازش داده‌ها

برای پیش‌پردازش دادگان، از کتابخانه re که از regex استفاده می‌کند، بهره می‌بریم. مراحل پیش‌پردازش به ترتیب در زیر بیان شده و در مورد هر کدام توضیحی داده می‌شود و نمونه‌ای روی آن اجرا می‌شود. نمونه‌ای که قرار است به عنوان مثال بررسی کنیم، توئیت زیر است:

* پس اون 1500 نفری که در #آبان_۹۸ کشتین فقط بخاطر اینکه اعتراض مسالمت آمیز که از حقوق اولیه مردمه و در قانون اساسی هم اومده آدم محترم نبودن؟ اون بیچاره هایی که در #هواپیمای_اوکراینی کشتین در حالیکه بدون کوچکترین آزاری در حال خروج قانونی از ایران بودن چی؟ اونها محترم نبودن؟

1. در مرحله اول از تابع remove_html_and_urls که تعریف کردیم استفاده می‌کنیم تا html و urlهایی که در توئیت وجود دارند را حذف کنیم. دلیل وجود html و url ها در این دیتاست، همانطور که مقاله گفته این است که هنگام scrap کردن این توئیت‌ها، چون ویدئو و یا تصاویر نیز وجود داشتند، بنابراین آدرس آن‌ها به عنوان مثال scrap شده.

   پس اون 1500 نفری که در #آبان_۹۸ کشتین فقط بخاطر اینکه اعتراض مسالمت آمیز که از حقوق اولیه مردمه و در قانون اساسی هم اومده آدم محترم نبودن؟ اون بیچاره هایی که در #هواپیمای_اوکراینی کشتین در حالیکه بدون کوچکترین آزاری در حال خروج قانونی از ایران بودن چی؟ اونها محترم نبودن؟

2. در مرحله بعدی نقطه گذاری‌ها را حذف می‌کنیم. به این صورت که تمام علائم کمکی از قبیل نقطه، ویرگول، علامت تعجب و ... را حذف می‌کنیم چرا که در فرآیند آموزش و تشخیص احساسات متن به ما کمکی نخواهند کرد.

   پس اون 1500 نفری که در آبان_۹۸ کشتین فقط بخاطر اینکه اعتراض مسالمت آمیز که از حقوق اولیه مردمه و در قانون اساسی هم اومده آدم محترم نبودن اون بیچاره هایی که در هواپیمای_اوکراینی کشتین در حالیکه بدون کوچکترین آزاری در حال خروج قانونی از ایران بودن چی اونها محترم نبودن

3. در مرحله سوم کاراکترهایی که از کاراکترهای فارسی نیستند را حذف می‌کنیم. چون برای مثال کاراکترهای چینی و یا کلماتی به زبان انگلیسی نیز در توئیت‌ها وجود دارند. کار بهتری که می‌توان در این مورد انجام داد این است که آن‌ها را با ترجمه آن‌ها جایگزین کنیم که این‌کار نیازمند یک دیکشنری نسبتا بزرگ خواهد بود.

   پس اون نفری که در آبان۹۸ کشتین فقط بخاطر اینکه اعتراض مسالمت آمیز که از حقوق اولیه مردمه و در قانون اساسی هم اومده آدم محترم نبودن اون بیچاره هایی که در هواپیمایاوکراینی کشتین در حالیکه بدون کوچکترین آزاری در حال خروج قانونی از ایران بودن چی اونها محترم نبودن

4. در مرحله چهارم stop word ها را حذف می‌کنیم. این کلمات در واقع کلمات پرتکراری هستند که در معنای جمله تاثیر خاصی نداشته و می‌توانند حذف شوند. stop wordها کلماتی مانند از، در، به و ... هستند.

اون نفری آبان ۹۸ کشتین بخاطر اعتراض مسالمت آمیز حقوق اولیه مردمه قانون اساسی اومده آدم محترم نبودن اون بیچاره هواپیمایاوکراینی کشتین حالیکه کوچکترین آزاری خروج قانونی ایران چی اونها محترم نبودن

5. سپس حروفی که در یک کلمه بیشتر از حد معلول (نهایت ۲ بار حد معمول در نظر گرفته شده) وجود داشته باشند را حذف کرده و با یکی جایگزین می‌کند.

اون نفری آبان ۹۸ کشتین بخاطر اعتراض مسالمت آمیز حقوق اولیه مردمه قانون اساسی اومده آدم محترم نبودن اون بیچاره هواپیمایاوکراینی کشتین حالیکه کوچکترین آزاری خروج قانونی ایران چی اونها محترم نبودن

6. حال emojiها را با کلمات نظیر آن‌ها در فارسی جایگزین می‌کنیم. روش دیگر این بود که emoji ها را به متن انگلیسی تبدیل کنیم و بعد دوباره با استفاده از دیکشنری به فارسی ترجمه کنیم.

اون نفری آبان ۹۸ کشتین بخاطر اعتراض مسالمت آمیز حقوق اولیه مردمه قانون اساسی اومده آدم محترم نبودن اون بیچاره هواپیمایاوکراینی کشتین حالیکه کوچکترین آزاری خروج قانونی ایران چی اونها محترم نبودن

7. حال در نهایت ریشه کلمات را جایگزین آن‌ها می‌کنیم تا در این صورت کلاس‌بندی و تشخیص کلمات مشابه، یکسان شده و تفاوتی مثلا برای یک کلمه و جمع آن، قائل نشویم.

اون نفر آب ۹۸ کشتین بخاطر اعتراض مسالم آمیز حقوق اولیه مردمه قانون اساس اومده آد محتر نبودن اون بیچاره هواپیمایاوکراین کشتین حالیکه کوچک آزار خروج قانون ایر چ اون محتر نبودن

۳-۱. نمایش ویژگی

برای استفاده از مدل‌هایی برای انجام مسئله طبقه‌بندی بر روی توئیت‌ها، نمی‌توانیم از خود کلمات استفاده کنیم و نیاز داریم که از یک بردار یا عدد به جای خود کلمات استفاده کنیم. می‌توانیم از توکن‌ساز مدل ParsBERT استفاده کنیم تا کلمات داخل هر رشته (جمله پردازش شده برای استفاده در مدل) را به یک ID تبدیل کند. برای استفاده از این tokenizer، از مدل bert-base-parsbert-uncased که توسط آزمایشگاه HooshvareLab آموزش دیده، استفاده می‌کنیم تا بتوانیم به هر کلمه (در واقع بعد از پردازش اولیه، کلمات ما یک‌سری کاراکتر به هم چسبیده و معمولا بی‌معنا خواهند بود)

این کار را در قسمت Dataloader انجام می‌دهیم. در واقع برای استفاده از داده‌ها، و وارد کردن آن‌ها به مدل، از Dataloader در پایتورچ استفاده می‌کنیم. این کلاس که دیتا مدل را تامین می‌کند، سه متد دارد که به ترتیب به بررسی آن‌ها خواهیم پرداخت.

- __init__: در این بخش مقادیر اولیه را بستگی به خواست خود و مدلی که داریم، مقداردهی می‌کنیم. همچنین در این بخش tokenizer خود را از مدل از پیش آموزش دیده ParsBert، ست می‌کنیم.
- __len__: این متد تنها تعداد داده‌ها را نشان می‌دهد.
- __getitem__: در این متد، توئیت‌ها با استفاده از tokenizer و با در نظر گرفتن طول ۳۲ و لایه گذاری (padding) تا همه آن‌ها در صورت کوتاه تر بودن، یک اندازه باشند، به آرایه‌ای از اعداد به طول ۳۲ تبدیل خواهند شد. سپس داده مورد نیاز و ID شده را برمی‌گردانیم.

سپس داده‌های خود را طبق خواسته مسئله ابتدا به نسبت ۷۰-۳۰ به مجموعه آموزش و تست تقسیم می‌کنیم. بعد از آن دوباره مجموعه آموزش را به نسبت ۸۰ به ۲۰، به مجموعه آموزش و اعتبارسنجی تقسیم خواهیم کرد.

بعد از آن دیتالودرهای مد نظر برای هر کدام از مجموعه‌های آموزش، اعتبارسنجی و تست را برای هر مدل با پارامترهای مختلف خواهیم ساخت و از آن‌ها استفاده خواهیم کرد. (برای مثال batch_size ممکن است تغییر کند)

همچنین می‌دانیم که برای آموزش مدل، نیاز داریم تا هر کدام از کلمات داخل جمله را به جای نمایش با یک ID، با یک آرایه نمایش دهیم. به این آرایه، تعبیه و یا embedding گفته می‌شود. یکی از خاصیت‌های مهم این embeddingها این است که کلمات شبیه و نزدیک به هم، بردارهای شبیه به هم‌تری دارند. و یا می‌توان گفت که فواصل این بردارها، نسبت به مقایسه آن‌ها با بقیه، کم‌تر است.

ما این کار را در قسمت مدل و قبل از ورودی دادن به بخش اصلی به کمک مدل از پیش آموزش دیده ParsBERT، انجام می‌دهیم. در واقع در ابتدا آرایه‌ای از IDها را ورودی داده و سپس به کمک این مدل، خروجی embedding آن‌ها را دریافت می‌کنیم و در ادامه از این embedding برای ورودی به سایر بخش‌ها (مدل اصلی در واقع) استفاده می‌کنیم.

حال در مسئله طرح شده از ما خواسته شده تا به گونه‌ای از این مدل ParsBERT استفاده کنیم که طول تعبیه یا embeddingهای بدست آمده برابر ۱۲۰ باشد. یکی از ایده‌های اولیه این است که یک لایه Dense یا در واقع fully connected بعد از آن استفاده کنیم تا طول تعبیه‌های تعریف شده برای ParsBERT (که برابر ۷۶۸ است) را به طول دلخواه (۱۲۰) تبدیل کنیم. اما ایراد این کار این است که با ایجاد یک لایه جدید، وزن‌های جدیدی به وجود آمده و این تعبیه‌ها با طول ۱۲۰ متاثر از این وزن‌های کاملا تصادفی خواهند بود. بنابراین تعبیه‌های بدست آمده ممکن است (با احتمال بسیار زیاد) که اصلا هیچ معنایی و همچنین ویژگی‌هایی که ما انتظار داریم را نداشته باشند. در واقع به گونه‌ای تاثیر آموزش زیاد ParsBERT روی داده‌های فراوان را از بین می‌برد و گویی تنها یک بردار با طول ۱۲۰ بدون هیچ مشخصه‌ای خواهد داد. حتی با وجود امکان آموزش برای این لایه، باز هم ویژگی مد نظر که ParsBERT آن را تامین می‌کرد، بدست نخواهد آمد.

روش اصلی انجام این کار، تغییر پیکربندی خود مدل ParsBERT است. برای اینکار، از روش زیر استفاده کردم.

```
config = AutoConfig.from_pretrained("HooshvareLab/bert-base-parsbert-uncased")
self.bert = AutoModel.from_pretrained("HooshvareLab/bert-base-parsbert-uncased", config=config)

# Adjusting token embeddings, position embeddings, and token type embeddings to the required embedding size
self.bert.embeddings.word_embeddings = nn.Embedding(config.vocab_size, embedding_size)
self.bert.embeddings.position_embeddings = nn.Embedding(config.max_position_embeddings, embedding_size)
self.bert.embeddings.token_type_embeddings = nn.Embedding(config.type_vocab_size, embedding_size)
self.bert.embeddings.LayerNorm = nn.LayerNorm(embedding_size, eps=config.layer_norm_eps)
```

شکل 2. تغییر پیکربندی **ParsBERT**

اما با تلاش و جست و جوی زیاد، نتوانستم مشکلی که به آن برمی‌خوردم را پیدا و برطرف کنم. تعدادی از آن‌ها را درست کردم اما باز هم ارورهایی که نیازمند تغییر بیشتر پیکربندی بودند میخوردم که نمی‌دانستم آن‌ها را چطور برطرف کنم.

بنابراین از خود طول تعبیه پیش فرض در مدل ParsBERT استفاده کردم. طول تعبیه پیش فرض در این مدل، برابر ۷۶۸ است. دلیل این اتفاق ریشه در ساختار خود این مدل است. در این مدل هر کلمه در متن ورودی، به یک بردار تبدیل می‌شود که به آن تعبیه می‌گویند. این همانطور که در صورت مسئله گفته شده، این تعبیه‌ها در واقع احساسات و عواطف (معنای واقعی) منتقل شده از هر کلمه درون این متن را بیان می‌کنند به گونه‌ای که قابل فهم برای مدل باشد.

در این مدل از تعدادی لایه ترنسفورمری استفاده شده. هر لایه تعداد مشخصی self-attention دارد. وقتی یک نمونه به آن ورودی می‌دهیم، از هرکدام از این لایه‌ها عبور کرده و بعد از عبور از آن، یک وکتور با طول خاص آن لایه تولید می‌کند به بیانگر معنا در آن ابعاد است. همچنین همانطور که قبلا اشاره کردیم، این مفاهیم و معنا کلمات وابسته به جمله پیدا خواهند شد و تک بعدی (مانند یک دیکشنری) عمل نمی‌کنند. اگر این اتفاق نمی‌افتاد برای کلماتی مانند «شیر»، دچار مشکل می‌شدیم.

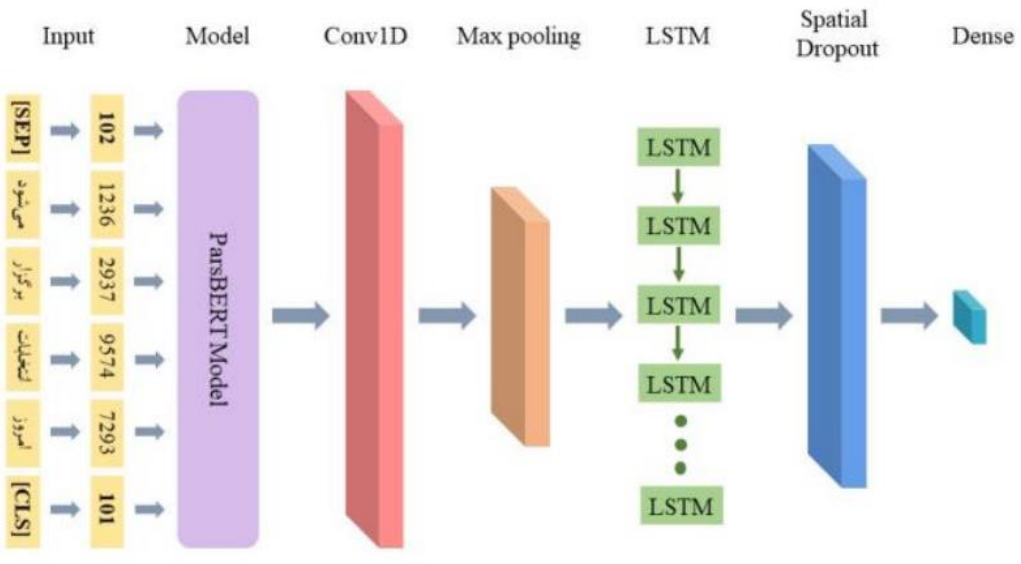
حال دلیل انتخاب ۷۶۸ به عنوان طول تعبیه‌های پیش فرض برای این مدل، تا حدودی دلخواه هست. اما از نظر تجربی، برای طیف وسیعی از وظایف پردازش زبان طبیعی (NLP) مناسب و موثر است. هر بعد و در واقع هر کدام از این ۷۶۸ مقدار را می‌توان نمایاگر یک ویژگی دید که در جمله ورودی دیده می‌شود. همچنین ما توانایی درک این ویژگی‌ها و تفسیر آن‌ها را نداریم و مدل در طول فرآیند آموزش آن‌ها را یاد می‌گیرد. با این حال الگوهای زبانی و معناشناسی مهمی را به تصویر می‌کشند که در وظایف مختلف پردازش زبان طبیعی بسیار کمک کننده هستند.

۱-۴. ساخت مدل

همانطور که در بخش داده و ساخت Dataloader گفتیم، دیتاست موجود و پیش پردازش شده طبق خواسته سوال ابتدا به نسبت ۷۰-۳۰ برای آموزش و تست و سپس داده‌های آموزش به نسبت ۸۰-۲۰ به داده‌های نهایی آموزش و اعتبارسنجی تقسیم شده‌اند.

۱-۴-۱. CNN-LSTM

حال به بررسی مدل ترکیبی استفاده شده در مقاله می‌پردازیم. این مدل از یک شبکه CNN و سپس یک شبکه LSTM استفاده کرده تا این مسئله classification را انجام دهد. همانگونه که در مقاله نیز بیان شده، دلیل این ایده این است که اولا CNN توانایی بالایی در پیدا کردن ویژگی‌ها (Feature Extraction) دارد. بنابراین می‌توانیم ویژگی‌ها را در مجموعه دادگان به این شکل (مانند متن، تصویر و ...) به نحو احسنت بدست آوریم. از طرفی ایده استفاده از LSTM به این دلیل است که این شبکه توانایی بالایی در به خاطر سپردن دارد، بنابراین می‌توانیم با استفاده از آن، یک طبقه‌بندی خوب روی جمله‌ها (توئیت‌ها) انجام دهیم، چرا که تمام جمله با استفاده از attentionها به خاطر سپرده و از آن‌ها در تسک مورد نظر بهره می‌برد. شمای کلی این مدل نیز به این صورت است:



شکل 3. ساختار شبکه **CNN-LSTM**

این شبکه را دقیقا به صورت شکل ۳، پیاده‌سازی می‌کنیم. کلاس آن ۲ متد دارد که به بررسی هر کدام می‌پردازیم.

- در ابتدا همانطور که قبلا نیز اشاره کردیم، مدل BERT خود را که در اینجا همان ParsBERT است، تعریف می‌کنیم. سپس همانطور که از شکل ۳ مشخص است، از یک لایه کانولوشنی یک بعدی عبور می‌دهیم و بعد از آن از Max Pooling استفاده می‌کنیم تا سایز Feature Mapها کمی کوچک‌تر شود. حال زمان استفاده از LSTM است. که آن را با ورودی ۳۲ تعریف می‌کنیم. بعد از آن از Spatial DropOut استفاده می‌کنیم. در نهایت نیز برای classify کردن داده‌ها، نیاز داریم که از یک لایه fully connected (Dense) استفاده کنیم که به تعداد کلاس‌ها (در اینجا ۶ کلاس داریم) نورون دارد.
- forward: در این متد نیز به ترتیب ورودی را به لایه‌ها تعریف شده در قسمت قبل می‌دهیم. تنها نکته این است که ورودی را بدون محاسبه گرادیان و بروز رسانی از مدل BERT عبور می‌دهیم.

حال همانطور که از ما خواسته شده، مدل را با پارامترهای مختلف (در کل ۸ حالت آموزش می‌دهیم). این پارامترها به صورت زیر هستند:

- Batch Size = [8, 64]
- Learning Rate = [0.001, 0.0001]
- Optimizers = [Adam, SGD]

حال نتایج بدست آمده از هر کدام را بررسی خواهیم کرد.

1. Batch_size = 8, Learning_rate = 0.001, Optimizer = Adam

```
Initializing model with batch_size=8, lr=0.001, optimizer=Adam
Epoch 1/10: Train Accuracy: 0.57, Train Precision: 0.60, Train Recall: 0.57, Train F1: 0.56
Val Accuracy: 0.66, Val Precision: 0.72, Val Recall: 0.66, Val F1: 0.66
Epoch 2/10: Train Accuracy: 0.68, Train Precision: 0.71, Train Recall: 0.68, Train F1: 0.68
Val Accuracy: 0.69, Val Precision: 0.73, Val Recall: 0.69, Val F1: 0.69
Epoch 3/10: Train Accuracy: 0.70, Train Precision: 0.73, Train Recall: 0.70, Train F1: 0.70
Val Accuracy: 0.70, Val Precision: 0.74, Val Recall: 0.70, Val F1: 0.70
Epoch 4/10: Train Accuracy: 0.71, Train Precision: 0.74, Train Recall: 0.71, Train F1: 0.71
Val Accuracy: 0.71, Val Precision: 0.75, Val Recall: 0.71, Val F1: 0.71
Epoch 5/10: Train Accuracy: 0.72, Train Precision: 0.75, Train Recall: 0.72, Train F1: 0.72
Val Accuracy: 0.71, Val Precision: 0.75, Val Recall: 0.71, Val F1: 0.71
Epoch 6/10: Train Accuracy: 0.72, Train Precision: 0.75, Train Recall: 0.72, Train F1: 0.72
Val Accuracy: 0.71, Val Precision: 0.75, Val Recall: 0.71, Val F1: 0.71
Epoch 7/10: Train Accuracy: 0.72, Train Precision: 0.76, Train Recall: 0.72, Train F1: 0.72
Val Accuracy: 0.71, Val Precision: 0.74, Val Recall: 0.71, Val F1: 0.71
Epoch 8/10: Train Accuracy: 0.73, Train Precision: 0.76, Train Recall: 0.73, Train F1: 0.73
Val Accuracy: 0.72, Val Precision: 0.76, Val Recall: 0.72, Val F1: 0.72
Epoch 9/10: Train Accuracy: 0.73, Train Precision: 0.76, Train Recall: 0.73, Train F1: 0.73
Val Accuracy: 0.71, Val Precision: 0.75, Val Recall: 0.71, Val F1: 0.71
```

Epoch 10/10: Train Accuracy: 0.73, Train Precision: 0.76, Train Recall: 0.73, Train F1: 0.73
Val Accuracy: 0.72, Val Precision: 0.76, Val Recall: 0.72, Val F1: 0.72
Test Accuracy: 0.72, Test Precision: 0.76, Test Recall: 0.72, Test F1: 0.72
Initializing model with batch_size=8, lr=0.001, optimizer=SGD

Batch_size = 8, Learning_rate = 0.001, Optimizer = SGD .2

Initializing model with batch_size=8, lr=0.001, optimizer=SGD
Epoch 1/10: Train Accuracy: 0.31, Train Precision: 0.29, Train Recall: 0.31, Train F1: 0.19
Val Accuracy: 0.37, Val Precision: 0.23, Val Recall: 0.37, Val F1: 0.26
Epoch 2/10: Train Accuracy: 0.39, Train Precision: 0.40, Train Recall: 0.39, Train F1: 0.31
Val Accuracy: 0.43, Val Precision: 0.44, Val Recall: 0.43, Val F1: 0.39
Epoch 3/10: Train Accuracy: 0.45, Train Precision: 0.45, Train Recall: 0.45, Train F1: 0.41
Val Accuracy: 0.46, Val Precision: 0.51, Val Recall: 0.46, Val F1: 0.41
Epoch 4/10: Train Accuracy: 0.48, Train Precision: 0.49, Train Recall: 0.48, Train F1: 0.46
Val Accuracy: 0.50, Val Precision: 0.51, Val Recall: 0.50, Val F1: 0.48
Epoch 5/10: Train Accuracy: 0.50, Train Precision: 0.51, Train Recall: 0.50, Train F1: 0.49
Val Accuracy: 0.52, Val Precision: 0.56, Val Recall: 0.52, Val F1: 0.51
Epoch 6/10: Train Accuracy: 0.52, Train Precision: 0.53, Train Recall: 0.52, Train F1: 0.51
Val Accuracy: 0.52, Val Precision: 0.57, Val Recall: 0.52, Val F1: 0.51
Epoch 7/10: Train Accuracy: 0.53, Train Precision: 0.54, Train Recall: 0.53, Train F1: 0.52
Val Accuracy: 0.53, Val Precision: 0.56, Val Recall: 0.53, Val F1: 0.52
Epoch 8/10: Train Accuracy: 0.53, Train Precision: 0.55, Train Recall: 0.53, Train F1: 0.52
Val Accuracy: 0.54, Val Precision: 0.56, Val Recall: 0.54, Val F1: 0.54
Epoch 9/10: Train Accuracy: 0.54, Train Precision: 0.56, Train Recall: 0.54, Train F1: 0.53
Val Accuracy: 0.55, Val Precision: 0.58, Val Recall: 0.55, Val F1: 0.54
Epoch 10/10: Train Accuracy: 0.55, Train Precision: 0.56, Train Recall: 0.55, Train F1: 0.54
Val Accuracy: 0.55, Val Precision: 0.58, Val Recall: 0.55, Val F1: 0.53
Test Accuracy: 0.55, Test Precision: 0.57, Test Recall: 0.55, Test F1: 0.54

Batch_size = 8, Learning_rate = 0.0001, Optimizer = Adam .3

Initializing model with batch_size=8, lr=0.0001, optimizer=Adam
Epoch 1/10: Train Accuracy: 0.49, Train Precision: 0.51, Train Recall: 0.49, Train F1: 0.48
Val Accuracy: 0.53, Val Precision: 0.58, Val Recall: 0.53, Val F1: 0.52
Epoch 2/10: Train Accuracy: 0.55, Train Precision: 0.57, Train Recall: 0.55, Train F1: 0.54
Val Accuracy: 0.57, Val Precision: 0.59, Val Recall: 0.57, Val F1: 0.56
Epoch 3/10: Train Accuracy: 0.60, Train Precision: 0.63, Train Recall: 0.60, Train F1: 0.60
Val Accuracy: 0.62, Val Precision: 0.68, Val Recall: 0.62, Val F1: 0.62
Epoch 4/10: Train Accuracy: 0.64, Train Precision: 0.68, Train Recall: 0.64, Train F1: 0.64
Val Accuracy: 0.65, Val Precision: 0.69, Val Recall: 0.65, Val F1: 0.65
Epoch 5/10: Train Accuracy: 0.66, Train Precision: 0.70, Train Recall: 0.66, Train F1: 0.66
Val Accuracy: 0.67, Val Precision: 0.71, Val Recall: 0.67, Val F1: 0.67
Epoch 6/10: Train Accuracy: 0.68, Train Precision: 0.71, Train Recall: 0.68, Train F1: 0.68
Val Accuracy: 0.69, Val Precision: 0.72, Val Recall: 0.69, Val F1: 0.69
Epoch 7/10: Train Accuracy: 0.69, Train Precision: 0.73, Train Recall: 0.69, Train F1: 0.69
Val Accuracy: 0.69, Val Precision: 0.72, Val Recall: 0.69, Val F1: 0.69
Epoch 8/10: Train Accuracy: 0.70, Train Precision: 0.73, Train Recall: 0.70, Train F1: 0.70
Val Accuracy: 0.69, Val Precision: 0.73, Val Recall: 0.69, Val F1: 0.69
Epoch 9/10: Train Accuracy: 0.71, Train Precision: 0.74, Train Recall: 0.71, Train F1: 0.71
Val Accuracy: 0.69, Val Precision: 0.73, Val Recall: 0.69, Val F1: 0.69
Epoch 10/10: Train Accuracy: 0.71, Train Precision: 0.75, Train Recall: 0.71, Train F1: 0.71
Val Accuracy: 0.70, Val Precision: 0.74, Val Recall: 0.70, Val F1: 0.70
Test Accuracy: 0.70, Test Precision: 0.73, Test Recall: 0.70, Test F1: 0.70

Batch_size = 8, Learning_rate = 0.0001, Optimizer = SGD .4

Initializing model with batch_size=8, lr=0.0001, optimizer=SGD
Epoch 1/10: Train Accuracy: 0.24, Train Precision: 0.22, Train Recall: 0.24, Train F1: 0.20
Val Accuracy: 0.27, Val Precision: 0.37, Val Recall: 0.27, Val F1: 0.16
Epoch 2/10: Train Accuracy: 0.28, Train Precision: 0.12, Train Recall: 0.28, Train F1: 0.16
Val Accuracy: 0.27, Val Precision: 0.12, Val Recall: 0.27, Val F1: 0.16
Epoch 3/10: Train Accuracy: 0.28, Train Precision: 0.12, Train Recall: 0.28, Train F1: 0.16
Val Accuracy: 0.27, Val Precision: 0.12, Val Recall: 0.27, Val F1: 0.16
Epoch 4/10: Train Accuracy: 0.28, Train Precision: 0.37, Train Recall: 0.28, Train F1: 0.16
Val Accuracy: 0.28, Val Precision: 0.37, Val Recall: 0.28, Val F1: 0.16
Epoch 5/10: Train Accuracy: 0.30, Train Precision: 0.26, Train Recall: 0.30, Train F1: 0.14
Val Accuracy: 0.30, Val Precision: 0.25, Val Recall: 0.30, Val F1: 0.14
Epoch 6/10: Train Accuracy: 0.30, Train Precision: 0.30, Train Recall: 0.30, Train F1: 0.14
Val Accuracy: 0.30, Val Precision: 0.33, Val Recall: 0.30, Val F1: 0.14
Epoch 7/10: Train Accuracy: 0.30, Train Precision: 0.29, Train Recall: 0.30, Train F1: 0.15
Val Accuracy: 0.31, Val Precision: 0.29, Val Recall: 0.31, Val F1: 0.15
Epoch 8/10: Train Accuracy: 0.31, Train Precision: 0.29, Train Recall: 0.31, Train F1: 0.16
Val Accuracy: 0.31, Val Precision: 0.29, Val Recall: 0.31, Val F1: 0.16
Epoch 9/10: Train Accuracy: 0.32, Train Precision: 0.29, Train Recall: 0.32, Train F1: 0.18
Val Accuracy: 0.33, Val Precision: 0.27, Val Recall: 0.33, Val F1: 0.20
Epoch 10/10: Train Accuracy: 0.34, Train Precision: 0.26, Train Recall: 0.34, Train F1: 0.22
Val Accuracy: 0.34, Val Precision: 0.26, Val Recall: 0.34, Val F1: 0.22
Test Accuracy: 0.34, Test Precision: 0.26, Test Recall: 0.34, Test F1: 0.22

Batch_size = 64, Learning_rate = 0.001, Optimizer = Adam .5

Initializing model with batch_size=64, lr=0.001, optimizer=Adam
Epoch 1/10: Train Accuracy: 0.56, Train Precision: 0.59, Train Recall: 0.56, Train F1: 0.56
Val Accuracy: 0.63, Val Precision: 0.65, Val Recall: 0.63, Val F1: 0.63
Epoch 2/10: Train Accuracy: 0.67, Train Precision: 0.71, Train Recall: 0.67, Train F1: 0.67
Val Accuracy: 0.69, Val Precision: 0.73, Val Recall: 0.69, Val F1: 0.69
Epoch 3/10: Train Accuracy: 0.70, Train Precision: 0.73, Train Recall: 0.70, Train F1: 0.70
Val Accuracy: 0.70, Val Precision: 0.73, Val Recall: 0.70, Val F1: 0.70
Epoch 4/10: Train Accuracy: 0.71, Train Precision: 0.74, Train Recall: 0.71, Train F1: 0.71
Val Accuracy: 0.70, Val Precision: 0.74, Val Recall: 0.70, Val F1: 0.70
Epoch 5/10: Train Accuracy: 0.71, Train Precision: 0.75, Train Recall: 0.71, Train F1: 0.71
Val Accuracy: 0.71, Val Precision: 0.74, Val Recall: 0.71, Val F1: 0.71
Epoch 6/10: Train Accuracy: 0.72, Train Precision: 0.75, Train Recall: 0.72, Train F1: 0.72
Val Accuracy: 0.71, Val Precision: 0.75, Val Recall: 0.71, Val F1: 0.72
Epoch 7/10: Train Accuracy: 0.72, Train Precision: 0.76, Train Recall: 0.72, Train F1: 0.72
Val Accuracy: 0.71, Val Precision: 0.76, Val Recall: 0.71, Val F1: 0.72
Epoch 8/10: Train Accuracy: 0.72, Train Precision: 0.76, Train Recall: 0.72, Train F1: 0.72
Val Accuracy: 0.71, Val Precision: 0.75, Val Recall: 0.71, Val F1: 0.71
Epoch 9/10: Train Accuracy: 0.72, Train Precision: 0.76, Train Recall: 0.72, Train F1: 0.73
Val Accuracy: 0.72, Val Precision: 0.76, Val Recall: 0.72, Val F1: 0.72
Epoch 10/10: Train Accuracy: 0.72, Train Precision: 0.76, Train Recall: 0.72, Train F1: 0.73
Val Accuracy: 0.71, Val Precision: 0.77, Val Recall: 0.71, Val F1: 0.71
Test Accuracy: 0.71, Test Precision: 0.76, Test Recall: 0.71, Test F1: 0.71

Batch_size = 64, Learning_rate = 0.001, Optimizer = SGD .6

Initializing model with batch_size=64, lr=0.001, optimizer=SGD
Epoch 1/10: Train Accuracy: 0.31, Train Precision: 0.26, Train Recall: 0.31, Train F1: 0.19
Val Accuracy: 0.35, Val Precision: 0.27, Val Recall: 0.35, Val F1: 0.23
Epoch 2/10: Train Accuracy: 0.39, Train Precision: 0.46, Train Recall: 0.39, Train F1: 0.30
Val Accuracy: 0.42, Val Precision: 0.46, Val Recall: 0.42, Val F1: 0.36
Epoch 3/10: Train Accuracy: 0.46, Train Precision: 0.47, Train Recall: 0.46, Train F1: 0.43
Val Accuracy: 0.44, Val Precision: 0.51, Val Recall: 0.44, Val F1: 0.43
Epoch 4/10: Train Accuracy: 0.50, Train Precision: 0.51, Train Recall: 0.50, Train F1: 0.48
Val Accuracy: 0.51, Val Precision: 0.54, Val Recall: 0.51, Val F1: 0.51
Epoch 5/10: Train Accuracy: 0.52, Train Precision: 0.53, Train Recall: 0.52, Train F1: 0.51
Val Accuracy: 0.48, Val Precision: 0.56, Val Recall: 0.48, Val F1: 0.47
Epoch 6/10: Train Accuracy: 0.53, Train Precision: 0.54, Train Recall: 0.53, Train F1: 0.52
Val Accuracy: 0.50, Val Precision: 0.62, Val Recall: 0.50, Val F1: 0.48
Epoch 7/10: Train Accuracy: 0.53, Train Precision: 0.55, Train Recall: 0.53, Train F1: 0.52
Val Accuracy: 0.52, Val Precision: 0.56, Val Recall: 0.52, Val F1: 0.51
Epoch 8/10: Train Accuracy: 0.54, Train Precision: 0.56, Train Recall: 0.54, Train F1: 0.54
Val Accuracy: 0.54, Val Precision: 0.59, Val Recall: 0.54, Val F1: 0.54
Epoch 9/10: Train Accuracy: 0.54, Train Precision: 0.56, Train Recall: 0.54, Train F1: 0.54
Val Accuracy: 0.55, Val Precision: 0.57, Val Recall: 0.55, Val F1: 0.55
Epoch 10/10: Train Accuracy: 0.55, Train Precision: 0.57, Train Recall: 0.55, Train F1: 0.55
Val Accuracy: 0.55, Val Precision: 0.62, Val Recall: 0.55, Val F1: 0.54
Test Accuracy: 0.55, Test Precision: 0.62, Test Recall: 0.55, Test F1: 0.54

Batch_size = 64, Learning_rate = 0.0001, Optimizer = Adam .7

Initializing model with batch_size=64, lr=0.0001, optimizer=Adam
Epoch 1/10: Train Accuracy: 0.48, Train Precision: 0.51, Train Recall: 0.48, Train F1: 0.47
Val Accuracy: 0.53, Val Precision: 0.57, Val Recall: 0.53, Val F1: 0.52
Epoch 2/10: Train Accuracy: 0.54, Train Precision: 0.57, Train Recall: 0.54, Train F1: 0.54
Val Accuracy: 0.57, Val Precision: 0.63, Val Recall: 0.57, Val F1: 0.56
Epoch 3/10: Train Accuracy: 0.60, Train Precision: 0.62, Train Recall: 0.60, Train F1: 0.59
Val Accuracy: 0.62, Val Precision: 0.67, Val Recall: 0.62, Val F1: 0.62
Epoch 4/10: Train Accuracy: 0.64, Train Precision: 0.67, Train Recall: 0.64, Train F1: 0.63
Val Accuracy: 0.65, Val Precision: 0.69, Val Recall: 0.65, Val F1: 0.64
Epoch 5/10: Train Accuracy: 0.66, Train Precision: 0.69, Train Recall: 0.66, Train F1: 0.66
Val Accuracy: 0.67, Val Precision: 0.71, Val Recall: 0.67, Val F1: 0.67
Epoch 6/10: Train Accuracy: 0.68, Train Precision: 0.71, Train Recall: 0.68, Train F1: 0.68
Val Accuracy: 0.68, Val Precision: 0.72, Val Recall: 0.68, Val F1: 0.68
Epoch 7/10: Train Accuracy: 0.69, Train Precision: 0.72, Train Recall: 0.69, Train F1: 0.69
Val Accuracy: 0.69, Val Precision: 0.73, Val Recall: 0.69, Val F1: 0.69
Epoch 8/10: Train Accuracy: 0.70, Train Precision: 0.73, Train Recall: 0.70, Train F1: 0.70
Val Accuracy: 0.69, Val Precision: 0.74, Val Recall: 0.69, Val F1: 0.69
Epoch 9/10: Train Accuracy: 0.71, Train Precision: 0.74, Train Recall: 0.71, Train F1: 0.71
Val Accuracy: 0.70, Val Precision: 0.74, Val Recall: 0.70, Val F1: 0.70
Epoch 10/10: Train Accuracy: 0.72, Train Precision: 0.75, Train Recall: 0.72, Train F1: 0.72
Val Accuracy: 0.70, Val Precision: 0.74, Val Recall: 0.70, Val F1: 0.70
Test Accuracy: 0.70, Test Precision: 0.74, Test Recall: 0.70, Test F1: 0.70

8. Batch_size = 64, Learning_rate = 0.0001, Optimizer = SGD

```
Initializing model with batch_size=64, lr=0.0001, optimizer=SGD
Epoch 1/10: Train Accuracy: 0.30, Train Precision: 0.11, Train Recall: 0.30, Train F1: 0.14
Val Accuracy: 0.30, Val Precision: 0.09, Val Recall: 0.30, Val F1: 0.14
Epoch 2/10: Train Accuracy: 0.30, Train Precision: 0.09, Train Recall: 0.30, Train F1: 0.14
Val Accuracy: 0.30, Val Precision: 0.09, Val Recall: 0.30, Val F1: 0.14
Epoch 3/10: Train Accuracy: 0.30, Train Precision: 0.09, Train Recall: 0.30, Train F1: 0.14
Val Accuracy: 0.30, Val Precision: 0.09, Val Recall: 0.30, Val F1: 0.14
Epoch 4/10: Train Accuracy: 0.30, Train Precision: 0.09, Train Recall: 0.30, Train F1: 0.14
Val Accuracy: 0.30, Val Precision: 0.09, Val Recall: 0.30, Val F1: 0.14
Epoch 5/10: Train Accuracy: 0.30, Train Precision: 0.09, Train Recall: 0.30, Train F1: 0.14
Val Accuracy: 0.30, Val Precision: 0.09, Val Recall: 0.30, Val F1: 0.14
Epoch 6/10: Train Accuracy: 0.30, Train Precision: 0.09, Train Recall: 0.30, Train F1: 0.14
Val Accuracy: 0.30, Val Precision: 0.09, Val Recall: 0.30, Val F1: 0.14
Epoch 7/10: Train Accuracy: 0.30, Train Precision: 0.32, Train Recall: 0.30, Train F1: 0.14
Val Accuracy: 0.30, Val Precision: 0.34, Val Recall: 0.30, Val F1: 0.14
Epoch 8/10: Train Accuracy: 0.30, Train Precision: 0.33, Train Recall: 0.30, Train F1: 0.14
Val Accuracy: 0.30, Val Precision: 0.33, Val Recall: 0.30, Val F1: 0.15
Epoch 9/10: Train Accuracy: 0.31, Train Precision: 0.32, Train Recall: 0.31, Train F1: 0.16
Val Accuracy: 0.31, Val Precision: 0.32, Val Recall: 0.31, Val F1: 0.16
Epoch 10/10: Train Accuracy: 0.32, Train Precision: 0.30, Train Recall: 0.32, Train F1: 0.18
Val Accuracy: 0.32, Val Precision: 0.30, Val Recall: 0.32, Val F1: 0.19
Test Accuracy: 0.33, Test Precision: 0.30, Test Recall: 0.33, Test F1: 0.19
```

بنابراین بهترین نتایج با پارامترهای batch size = 8، learning rate = 0.001 و optimizer = Adam حاصل شد. حال مدل‌های CNN و LSTM جداگانه را با همین پارامترها آموزش می‌دهیم.

۱-۴-۲. CNN

حال یک مدل CNN را برای classification روی این داده‌ها آموزش می‌دهیم. اول از همه ساختار این شبکه را بررسی می‌کنیم. در این شبکه مانند حالت ادغامی CNN-LSTM، دوباره از مدل ParsBERT برای تبدیل کلمات به بردار استفاده می‌کنیم. این مدل در واقع همان مدل CNN-LSTM بدون لایه LSTM است. روند آموزش این شبکه با بهترین پارامترهای بدست آمده به این صورت است:

```
Epoch 1/10: Train Accuracy: 0.25, Train Precision: 0.25, Train Recall: 0.25, Train F1: 0.10
Val Accuracy: 0.25, Val Precision: 0.08, Val Recall: 0.25, Val F1: 0.10
Epoch 2/10: Train Accuracy: 0.25, Train Precision: 0.07, Train Recall: 0.25, Train F1: 0.10
Val Accuracy: 0.25, Val Precision: 0.07, Val Recall: 0.25, Val F1: 0.10
Epoch 3/10: Train Accuracy: 0.25, Train Precision: 0.07, Train Recall: 0.25, Train F1: 0.10
Val Accuracy: 0.25, Val Precision: 0.07, Val Recall: 0.25, Val F1: 0.10
Epoch 4/10: Train Accuracy: 0.25, Train Precision: 0.07, Train Recall: 0.25, Train F1: 0.10
Val Accuracy: 0.25, Val Precision: 0.08, Val Recall: 0.25, Val F1: 0.10
Epoch 5/10: Train Accuracy: 0.25, Train Precision: 0.07, Train Recall: 0.25, Train F1: 0.10
Val Accuracy: 0.25, Val Precision: 0.08, Val Recall: 0.25, Val F1: 0.10
Epoch 6/10: Train Accuracy: 0.25, Train Precision: 0.25, Train Recall: 0.25, Train F1: 0.10
Val Accuracy: 0.25, Val Precision: 0.08, Val Recall: 0.25, Val F1: 0.10
Epoch 7/10: Train Accuracy: 0.25, Train Precision: 0.07, Train Recall: 0.25, Train F1: 0.10
Val Accuracy: 0.25, Val Precision: 0.08, Val Recall: 0.25, Val F1: 0.10
Epoch 8/10: Train Accuracy: 0.25, Train Precision: 0.07, Train Recall: 0.25, Train F1: 0.10
Val Accuracy: 0.25, Val Precision: 0.08, Val Recall: 0.25, Val F1: 0.10
Epoch 9/10: Train Accuracy: 0.25, Train Precision: 0.07, Train Recall: 0.25, Train F1: 0.10
Val Accuracy: 0.25, Val Precision: 0.09, Val Recall: 0.25, Val F1: 0.10
Epoch 10/10: Train Accuracy: 0.25, Train Precision: 0.07, Train Recall: 0.25, Train F1: 0.10
Val Accuracy: 0.25, Val Precision: 0.08, Val Recall: 0.25, Val F1: 0.10
Test Accuracy: 0.24, Test Precision: 0.07, Test Recall: 0.24, Test F1: 0.10
```

حال به بررسی مزایا و معایب مدل CNN بپردازیم:

مزایا:

- **استخراج ویژگی‌های محلی** :شبکه‌های CNN در استخراج ویژگی‌های محلی مانندn-gram ها (ترکیب‌های کوتاه کلمات) در متن بسیار مؤثر هستند، که برای درک زمینه کوتاه مدت و اطلاعات محلی مفید است.
- **پارامترهای کمتر** :به دلیل اشتراک پارامترها در کانولوشن‌ها، شبکه‌های CNN نسبت به مدل‌های دیگر پارامترهای کمتری دارند که به کاهش هزینه محاسباتی و جلوگیری از بیش‌برازش کمک می‌کند.

- **کارایی محاسباتی** :شبکه‌های CNN به دلیل محاسبات موازی و بهره‌وری در استفاده از پردازنده‌های گرافیکی(GPU) ، سریع‌تر هستند.

معایب:

- **محدودیت در وابستگی‌های طولانی مدت** :شبکه‌های CNN در تشخیص وابستگی‌های طولانی مدت و ارتباطات بین کلمات دور در متن مشکل دارند، مگر اینکه لایه‌های عمیق‌تر و پیچیده‌تری اضافه شوند.
- **محدودیت در ترتیب کلمات** :شبکه‌های CNN ترتیب دقیق کلمات را در متن به خوبی مدل نمی‌کنند، که ممکن است درک معنایی دقیق را محدود کند.

۱-۴-۳. LSTM

این شبکه نیز همان مانند همان شبکه ترکیبی بوده و تنها تفاوت آن این است که Conv1D و max pooling حذف شده و ورودی‌ها مستقیم پس از عبور از ParsBERT و بدست آمدن تعبیه آن‌ها، وارد بخش LSTM می‌شوند. نتایج آموزش روی این مدل به این صورت است.

```
Epoch 1/10: Train Accuracy: 0.23, Train Precision: 0.21, Train Recall: 0.23, Train F1: 0.14
Val Accuracy: 0.22, Val Precision: 0.18, Val Recall: 0.22, Val F1: 0.14
Epoch 2/10: Train Accuracy: 0.23, Train Precision: 0.21, Train Recall: 0.23, Train F1: 0.14
Val Accuracy: 0.22, Val Precision: 0.24, Val Recall: 0.22, Val F1: 0.14
Epoch 3/10: Train Accuracy: 0.23, Train Precision: 0.18, Train Recall: 0.23, Train F1: 0.14
Val Accuracy: 0.22, Val Precision: 0.12, Val Recall: 0.22, Val F1: 0.14
Epoch 4/10: Train Accuracy: 0.23, Train Precision: 0.18, Train Recall: 0.23, Train F1: 0.14
Val Accuracy: 0.23, Val Precision: 0.13, Val Recall: 0.23, Val F1: 0.14
Epoch 5/10: Train Accuracy: 0.23, Train Precision: 0.20, Train Recall: 0.23, Train F1: 0.14
Val Accuracy: 0.23, Val Precision: 0.14, Val Recall: 0.23, Val F1: 0.14
Epoch 6/10: Train Accuracy: 0.23, Train Precision: 0.22, Train Recall: 0.23, Train F1: 0.14
Val Accuracy: 0.22, Val Precision: 0.18, Val Recall: 0.22, Val F1: 0.14
Epoch 7/10: Train Accuracy: 0.23, Train Precision: 0.19, Train Recall: 0.23, Train F1: 0.14
Val Accuracy: 0.23, Val Precision: 0.25, Val Recall: 0.23, Val F1: 0.14
Epoch 8/10: Train Accuracy: 0.23, Train Precision: 0.22, Train Recall: 0.23, Train F1: 0.14
Val Accuracy: 0.23, Val Precision: 0.35, Val Recall: 0.23, Val F1: 0.14
Epoch 9/10: Train Accuracy: 0.23, Train Precision: 0.20, Train Recall: 0.23, Train F1: 0.14
Val Accuracy: 0.23, Val Precision: 0.22, Val Recall: 0.23, Val F1: 0.14
Epoch 10/10: Train Accuracy: 0.23, Train Precision: 0.22, Train Recall: 0.23, Train F1: 0.14
Val Accuracy: 0.22, Val Precision: 0.20, Val Recall: 0.22, Val F1: 0.14
Test Accuracy: 0.23, Test Precision: 0.23, Test Recall: 0.23, Test F1: 0.14
```

حال به بررسی مزایا و معایب مدل LSTM بپردازیم:

مزایا:

- **مدل‌سازی وابستگی‌های طولانی مدت** :شبکه‌های LSTM برای مدل‌سازی وابستگی‌های طولانی مدت در متن‌ها بسیار مناسب هستند، که برای درک متن‌هایی با جملات بلند و پیچیده مفید است.
- **حافظه طولانی مدت** :شبکه‌های LSTM می‌توانند اطلاعات مهم را در طول توالی‌های طولانی نگه دارند، که برای وظایفی که به زمینه قبلی نیاز دارند بسیار مفید است.
- **انعطاف‌پذیری با طول توالی‌های متغیر** :شبکه‌های LSTM می‌توانند توالی‌هایی با طول‌های مختلف را پردازش کنند، که برای داده‌های متنی که ممکن است طول جملات متفاوتی داشته باشند، بسیار مناسب است.

معایب

- **پیچیدگی و زمان آموزش طولانی** :آموزش شبکه‌های LSTM به دلیل محاسبات تکراری و پیچیدگی بالا زمان‌بر است و ممکن است مشکلاتی مانند ناپدید شدن یا انفجار گرادیان رخ دهد.
- **هزینه محاسباتی بالا** :شبکه‌های LSTM از نظر محاسباتی سنگین‌تر هستند و به منابع سخت‌افزاری بیشتری نیاز دارند.
- **سختی در موازی‌سازی** :ماهیت تکراری شبکه‌های LSTM باعث می‌شود که پردازش موازی سخت باشد، که می‌تواند به زمان آموزش طولانی‌تر منجر شود.

حال به بررسی مزایا و معایب مدل ترکیبی این دو مدل می‌پردازیم.

مزایا:

- **ترکیب قوت‌ها** :این ترکیب از قدرت‌های هر دو شبکه بهره می‌برد — شبکه‌های CNN برای استخراج ویژگی‌های محلی از توالی‌های متنی و شبکه‌های LSTM برای مدل‌سازی وابستگی‌های طولانی مدت.

- **نمایش‌های غنی از ویژگی‌ها** :شبکه‌های CNN می‌توانند ویژگی‌های غنی و سطح بالایی از متن استخراج کنند، که شبکه LSTM می‌تواند برای درک الگوها و وابستگی‌های طولانی مدت استفاده کند.
- **کاربردپذیری چندگانه** :مدل‌های ترکیبی می‌توانند انواع مختلف وظایف NLP را انجام دهند، از جمله طبقه‌بندی متن، تحلیل احساسات و ترجمه ماشینی.

معایب

- **پیچیدگی بیشتر** :ترکیب شبکه‌های CNN و LSTM منجر به معماری‌های پیچیده‌تر می‌شود، که طراحی، تنظیم و اشکال‌زدایی آن‌ها دشوارتر است.
- **نیاز به منابع محاسباتی بیشتر** :ترکیب دو مدل سنگین می‌تواند به طور قابل توجهی نیازهای محاسباتی و استفاده از حافظه را افزایش دهد.
- **زمان آموزش طولانی‌تر** :آموزش مدل‌های ترکیبی معمولاً به دلیل افزایش تعداد پارامترها و پیچیدگی معماری، زمان بیشتری می‌برد.

در خلاصه، شبکه‌های CNN برای استخراج ویژگی‌های محلی و الگوهای کوتاه مدت در متن مؤثر هستند و برای داده‌های متنی کوتاه مانند توئیت‌ها مناسب‌اند. شبکه‌های LSTM برای مدل‌سازی وابستگی‌های طولانی مدت و فهم دقیق‌تر متون طولانی بهتر عمل می‌کنند. مدل‌های ترکیبی CNN-LSTM از هر دو نوع شبکه بهره می‌برند و ابزارهای قدرتمندی برای وظایفی که شامل ویژگی‌های محلی و طولانی مدت هستند فراهم می‌کنند، البته به هزینه افزایش پیچیدگی و نیازهای محاسباتی بیشتر. بنابراین اغدام این دو مدل با هدف کسب ویژگی‌های بهتر در عین وجود حافظه قوی‌تر، انجام می‌شود.

۱-۵. ارزیابی

	Accuracy	F1 score	Precision	Recall
CNN-LSTM	0.73	0.73	0.76	0.73
CNN	0.25	0.1	0.07	0.25
LSTM	0.23	0.22	0.14	0.23

جدول 1. جدول نتایج برای داده‌های آموزش

	Accuracy	F1 score	Precision	Recall
CNN-LSTM	0.72	0.72	0.76	0.72
CNN	0.25	0.1	0.08	0.25
LSTM	0.22	0.20	0.14	0.22

جدول 2. جدول نتایج برای داده‌های اعتبارسنجی

	Accuracy	F1 score	Precision	Recall
CNN-LSTM	0.72	0.72	0.76	0.72
CNN	0.24	0.1	0.07	0.24
LSTM	0.23	0.23	0.14	0.23

جدول 3. جدول نتایج برای داده‌های تست

بنابراین طبق نتایج بدست آمده، مشخص است که هدف به خوبی محقق شده و مدل ترکیبی همانطور که انتظار داشتیم، عملکرد بهتری دارد. علت عملکرد بد ۲ مدل دیگر تقریبا در نقاط ضعف و معایب هر کدام گفته شد. مثلا CNN ها عملکرد خوبی در مورد حافظه ندارند، بنابراین به خوبی نمی‌توانند یک جمله که از چندین کلمه تشکیل شده است را classify کنند. و برای LSTM، این مدل نمی‌تواند به خوبی CNN ویژگی‌ها را استخراج و از آن‌ها استفاده کند. بنابراین ترکیب آن‌ها عملکرد خیلی بهتری نسبت به هر کدام به تنهایی خواهد داشت.

حال به بررسی روش‌های مختلف برای محاسبه میانگین معیارهای ارزیابی می‌پردازیم. برای ارزیابی عملکرد مدل‌های یادگیری ماشین به ویژه در مسائل طبقه‌بندی، روش‌های مختلفی برای محاسبه میانگین معیارهای ارزیابی وجود دارد. این روش‌ها شامل averaging macro ، averaging micro و averaging weighted هستند. هر یک از این روش‌ها به شکل متفاوتی میانگین را محاسبه می‌کنند و تاثیرات مختلفی بر مقدار عددی معیارهای ارزیابی دارند.

- Averaging Macro:

در روش Averaging Macro، ابتدا معیارهای ارزیابی برای هر کلاس به طور جداگانه محاسبه می‌شوند و سپس میانگین این مقادیر گرفته می‌شود. این روش به تمام کلاس‌ها وزن مساوی می‌دهد و به همین دلیل برای مجموعه داده‌هایی که کلاس‌ها به طور متوازن توزیع نشده‌اند مناسب است.

تاثیر بر مسائل نامتوازن: در صورتی که داده‌ها نامتوازن باشند (یعنی تعداد نمونه‌های برخی کلاس‌ها بسیار بیشتر از بقیه باشد)، این روش می‌تواند عملکرد مدل را برای کلاس‌های کم‌تعداد بهتر نشان دهد، زیرا به هر کلاس وزن مساوی می‌دهد.

معایب: ممکن است عملکرد مدل را برای کلاس‌های بزرگ نادیده بگیرد و در نتیجه مقیاس دقیقی از عملکرد کلی مدل ارائه ندهد.

- Averaging Micro:

در روش averaging micro، ابتدا تمام پیش‌بینی‌های درست و نادرست برای تمام کلاس‌ها جمع‌آوری می‌شوند و سپس معیارهای ارزیابی محاسبه می‌شوند. این روش به هر نمونه وزن مساوی می‌دهد و بنابراین تاثیر کلاس‌های پرجمعیت بیشتر از کلاس‌های کم‌جمعیت است.

تاثیر بر مسائل نامتوازن: این روش به کلاس‌های پرجمعیت وزن بیشتری می‌دهد و بنابراین برای مجموعه داده‌های نامتوازن، معیارهای ارزیابی را به سمت کلاس‌های پرجمعیت می‌کشاند.

مزایا: در صورتی که هدف کلی شناسایی صحیح تمام نمونه‌ها باشد، این روش مناسب است زیرا تمام نمونه‌ها را به طور مساوی در نظر می‌گیرد.

- Averaging Weighted:

در روش averaging weighted، ابتدا معیارهای ارزیابی برای هر کلاس به طور جداگانه محاسبه می‌شوند، سپس این معیارها بر اساس تعداد نمونه‌های هر کلاس وزن‌دهی می‌شوند و در نهایت میانگین وزن‌دار محاسبه می‌شود. این روش ترکیبی از روش‌های micro و macro است و به هر کلاس بر اساس تعداد نمونه‌های آن وزن می‌دهد.

تاثیر بر مسائل نامتوازن: این روش به خوبی عملکرد مدل را در برابر مسائل نامتوازن نشان می‌دهد زیرا به کلاس‌های پرجمعیت وزن بیشتری می‌دهد، اما همچنان اطلاعات مربوط به کلاس‌های کم‌جمعیت را حفظ می‌کند.

مزایا: میانگین وزنی اطلاعات بیشتری نسبت به روش‌های macro و micro ارائه می‌دهد و می‌تواند تعادلی بین این دو روش ایجاد کند.

ما نیز در بررسی معیارهای ارزیابی و محاسبه میانگین آن‌ها، از روش Averaging Weighted استفاده کردیم.

پرسش ۲. سامانه‌های سایبرفیزیکی : نگهداری هوشمند

۲-۱. پیش‌پردازش داده‌ها

این مجموعه دادگان شامل ۱۰۰ موتور می‌باشد که هر سطر در این مجموعه، یک سری زمانی چند متغیره می‌باشد که هر یک از آنها متعلق به یک دستگاه خاص که همگی از یک نوع هستند اما هر موتور با درجات مختلفی از فرسودگی اولیه و تفاوت‌هایی در فرآیند تولید که ناشناخته و سالم در نظر گرفته می‌شوند، در نظر گرفته شده است که هر یک با unit number مشخص شده است. این مجموعه، به دو دسته‌ی آموزشی و ارزیابی تقسیم شده‌است. همانطور که می‌دانیم، هر سنسور پس از زمانی که شروع به کار می‌کند، ممکن است در قسمتی از مسیر دچار خطا شده و از کار بیفتد. در مجموعه‌ی آموزشی، اطلاعات مربوط به چرخه‌های هر دستگاه از زمانی که شروع به کار می‌کند تا زمانی که از کار بیفتد، ثبت شده است. همچنین، برای هر سری زمانی، شرایط عملیاتی نیز در سه ستون ثبت شده است که تغییر آنها روی عملکرد موتور اثرگذار است. همچنین، ۲۱ ستون تحت عنوان مقادیری که سنسورها برای هر موتور ثبت کرده‌اند، در نظر گرفته شده که نتیجه‌ی محاسبات سنسور را نشان می‌دهد. در زیر ساختار کلی متناظر با مجموعه دادگان آموزشی را مشاهده می‌کنیم:

	unit number	time in cycles	operational setting 1	operational setting 2	operational setting 3	sensor measurement 1	sensor measurement 2	sensor measurement 3	sensor measurement 4	sensor measurement 5	...	sensor measurement 12	sensor measurement 13	sensor measurement 14	sensor measurement 15	sensor measurement 16	sensor measurement 17	sensor measurement 18	sensor measurement 19	sensor measurement 20	sensor measurement 21
0	1	1	-0.0007	-0.0004	100.0	518.67	641.82	1589.70	1400.60	14.62	...	521.66	2388.02	8138.62	8.4195	0.03	392	2388	100.0	39.06	23.4190
1	1	2	0.0019	-0.0003	100.0	518.67	642.15	1591.82	1403.14	14.62	...	522.28	2388.07	8131.49	8.4318	0.03	392	2388	100.0	39.00	23.4236
2	1	3	-0.0043	0.0003	100.0	518.67	642.35	1587.99	1404.20	14.62	...	522.42	2388.03	8133.23	8.4178	0.03	390	2388	100.0	38.95	23.3442
3	1	4	0.0007	0.0000	100.0	518.67	642.35	1582.79	1401.87	14.62	...	522.86	2388.08	8133.83	8.3682	0.03	392	2388	100.0	38.88	23.3739
4	1	5	-0.0019	-0.0002	100.0	518.67	642.37	1582.85	1406.22	14.62	...	522.19	2388.04	8133.80	8.4294	0.03	393	2388	100.0	38.90	23.4044

شکل 4. ساختار مجموعه دادگان آموزشی

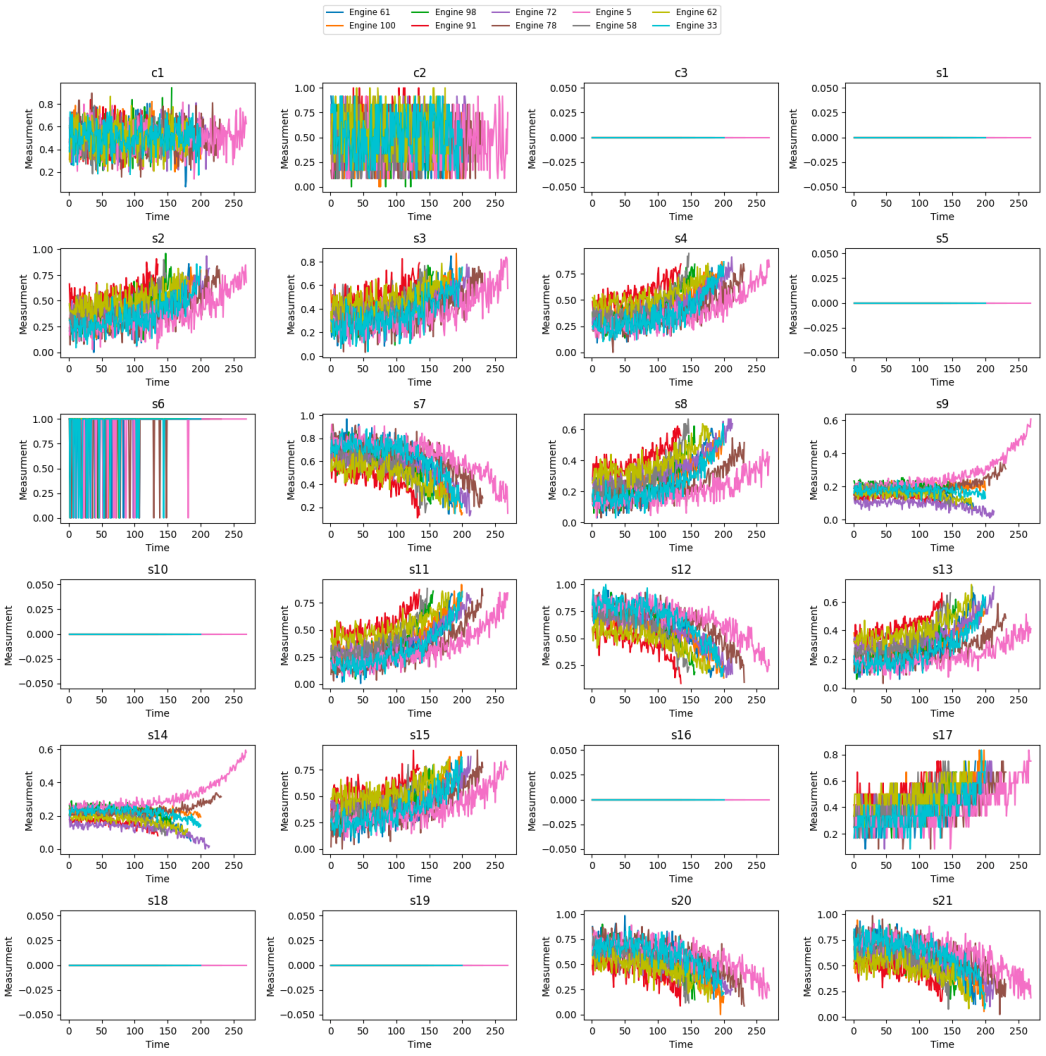
مجموعه‌ی آزمایشی نیز ساختاری مشابه دارد، با این تفاوت که این مجموعه برای همه‌ی موتورها تنها شامل بخشی از داده‌ها از زمانی که شروع به کار میکنند، تا قبل از خرابی آن را شامل می‌شود و هدف این است که عملکرد مدل پس از اتمام فرایند آموزش، با پیش‌بینی تعداد چرخه‌های عملیاتی باقی مانده پیش از خراب شدن روی مقادیر این داده‌ها ارزیابی شود.

برای انجام پیش پردازش روی داده‌های موجود، طبق مقاله، گام‌های زیر را طی می‌کنیم:

- **انتخاب داده‌ها**: در این مرحله، ستون‌هایی از دیتافریم که اطلاعات مفید برای حل مسئله را به ما نمی‌دهند، حذف می‌کنیم. همانطور که در شکل ۳ مشخص است، ستون‌های c3، s1، s5، s10، s16 و s19 اطلاعات خاصی منبی بر عملکرد موتورها به ما نمی‌دهند. بنابراین این ستون‌ها را حذف می‌کنیم تا با دیتای تمیزتری سر و کار داشته باشیم.
- **نرمال سازی**: برای اینکه کار با داده‌ها راحت‌تر باشد و محاسبات دقت بیشتری داشته باشد و مدل generalتر شود، نیاز است داده‌ها را در یک بازه، نرمال کنیم. برای این کار، از minMaxScaler متعلق به کتابخانه‌ی sklearn استفاده کرده‌ایم که با بکارگیری فرمول زیر، نرمال سازی را در بازه‌ی (0, 1) برای ما انجام می‌دهد:

$$x_i^* = \frac{x_i - x_{min}}{x_{max} - x_{min}}$$

معادله 1. فرمول نرمال‌سازی داده‌ها



شکل 5. نمایش مقادیر اندازه گرفته شده در سب زمان برای ۱۰ موتور تصادفی در مجموعه دادگان آموزشی

- **برچسب گذاری داده‌ها**: برای انجام تسک‌های regression و classification نیاز است داده‌ها را برچسب گذاری کنیم. برای انجام این‌کار و با در نظر گرفتن هر یک از این دو تسک داریم:

Regression: برای هر یک از داده‌ها در مجموعه‌ی آموزشی، مقدار متناظر با بیشترین زمانی که یک موتور کار کرده و پس از آن از کار افتاده است را در نظر میگیریم و زمان کارایی فعلی آن عضو از دادگان آموزشی را از آن کم می‌کنیم. به این صورت مقدار RUL را حساب کرده ایم. این مقدار را برای هر عضو مجموعه‌ی آموزشی محاسبه می‌کنیم و در یک ستون جدید در دیتافریم قرار می‌دهیم. به این ترتیب، برای انحام رگرشن، اده‌ها را لیبل‌گذاری می‌کنیم. در ادامه، فرمول مورد نظر برای محاسبه‌ی این مقدار آورده شده است:

$$RUL^{(i)}(t) = T^{(i)} - t^{(i)}, T^{(i)} > t^{(i)}$$

معادله 2. فرمول محاسبه‌ی**RUL**

در این شرایط، RUL به صورت خطی با گذر زمان کاهش می‌یابد و هرچه به پایان عمر موتور نزدیک می‌شویم، مقدار آن کمتر میشود.

Classification: برای طبقه بندی دودویی، لازم است یک آستانه تعریف کرده و برای هر مورد، تعیین کنیم داده‌ی مورد نظر نسبت به این آستانه، کجا قرار دارد. به این صورت دادگان را به دو کلاس طبقه بندی کنیم. با استفاده از مقدار محاسبه شده در قسمت قبل برای RUL هر یک از اعضای مجموعه، یک آستانه که در مقاله 50 در نظر گرفته شده‌است را در نظر میگیریم. نمونه‌هایی که RUL متناظر با آنها بیشتر از این آستانه باشد را با 1 برچسب گذاری می‌کنیم وآن‌ها را به عنوان سالم در نظر می‌گیریم. سایر موارد را به عنوان معیوب و با 0 برچسب گذاری می‌کنیم. در جدول زیر تعداد داده‌ها در هر یک از دو دسته آمده است:

Class 0(faulty condition)	Class 1(healthy condition)
5100	12631

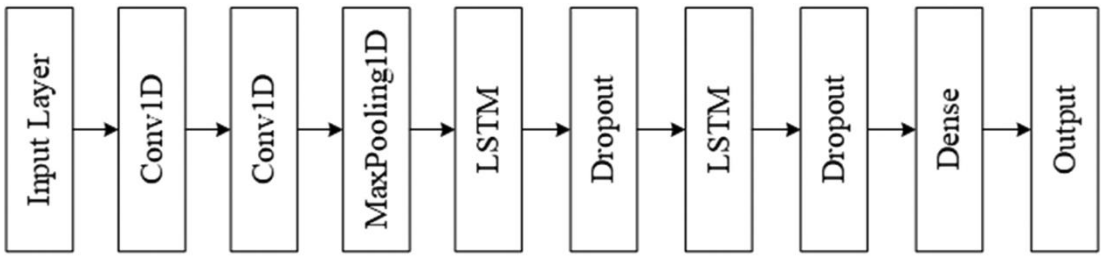
جدول 4. تعداد داده‌ها در هر یک از کلاس‌های سالم و معیوب

- **پردازشTime-window :** به عنوان یکی از گام‌های پیش‌پردازش داده نیاز است داده‌ها را در هر دو دسته‌ی آموزشی و آزمایشی به پنجره‌های زمانی تقسیم کنیم که در مقاله اندازه‌ی این پنجره ها ۳۰ در نظر گرفته شده است. این مرحله را نیز روی دیتا اعمال خواهیم کرد تا در اینده از ان برای مسئله‌ی regression استفاده کنیم طوریکه برای هر موتور دو حالت را بررسی خواهیم کرد، حالتی که تمام پنجره ها در نظر گرفته شوند و حالتی که تنها پنجره‌ی اخر برای هر موتور در نظر گرفته شود.

۲-۲. مدل‌سازی و ارزیابی

Classification:

با توجه به ساهتاری که در مقاله برای مدل ارائه شده، مدل را مب‌سازیم. شکل کلی این مدل به صورت زیر است.



شکل 6. ساختار مدل برای پیش‌بینی **RUL**

در جدول زیر به تفصیل معماری مدل پیاده شده برای مسئلهی طبقه‌بندی ارائه شده است:

Layer Index	Type	Filters/ Neurons	Filter Size	Region	Activation function	Dropout rate
1	Conv1D	32	5	-	ReLu	-
2	Conv1D	64	3	-	ReLu	-
3	MaxPooling1D	-	-	3	-	-
4	LSTM	50	-	-	Tanh	0.2
5	LSTM	50	-	-	Tanh	0.2
6	Dense (classification)	1	-	-	sigmoid	-
7	Dense (Regression)	1	-	-	linear	-

جدول 5. هایپرپارامترهای شبکه‌ی **CNN-LSTM**

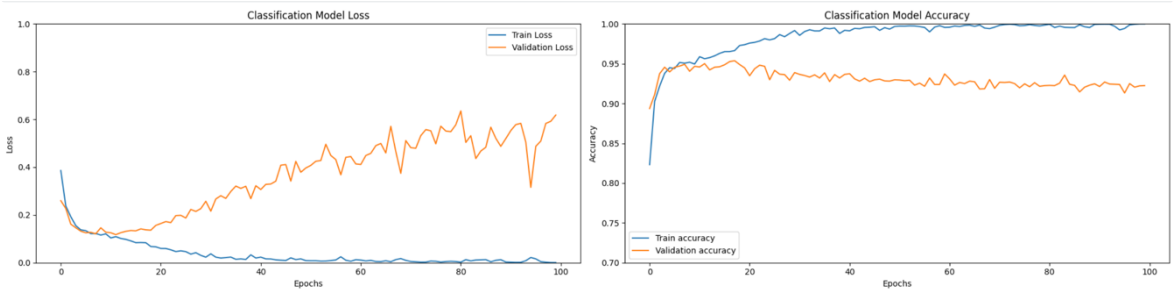
همانطور که گفته شده، برای بخش طبقه بندی، از:

- فعال‌سازی adam،
- تابع هزینه‌ی binary cross entropy
- تعداد 100 و
- Batch size مساوی 200

استفاده می‌کنیم.

لازم به ذکر است که 20% از داده‌های آموزشی را به عنوان داده‌ی اعتبارسنجی برای بررسی عملکرد مدل حین آموزش استفاده کرده‌ایم.

ابتدا حالتی را بررسی می‌کنیم که از early stopping استفاده نکرده‌ایم و تمام 100 ایپاک به طول کامل انجام می‌شوند. در این حالت نمودار تغییرات loss و accuracy به شکل زیر است:



شکل 7. تغییرات **loss** و **accuracy** حین آموزش مدل بدون در نظر گرفتن **early stopping**

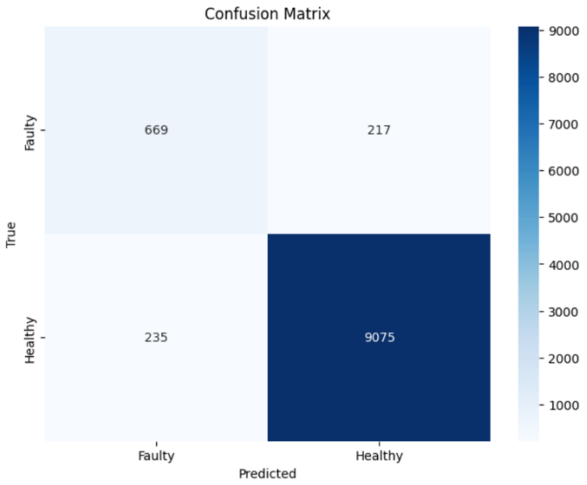
همانطور که از شکل بالا مشخص است، در فرایند آموزش مدل، تا حدود ایپاک ۲۰ هم روی داده‌های آموزشی و هم روی داده‌های اعتبارسنجی loss و accuracy به ترتیب کاهش و افزایش می‌یابند. پس از آن، loss و accuracy روی داده‌های آموزشی به ترتیب همچنان روند نزولی و صعودی دارند. اما loss روی داده‌های اعتبارسنجی افزایش می‌یابد درحالیکه accuracy تغییر چندانی نمی‌کند و در یک بازه، نوسان دارد.

با ارزیابی این مدل روی داده‌های تست به نتایج زیر دست پیدا کرده‌ایم:

Accuracy	Precision	Recall	F1-Score
95.6%	**97.7%**	**97.5%**	**97.6%**

جدول 6. نتایج ارزیابی مدل روی کل داده‌های تست

ماتریس درهم ریختگی متناظر با عملکرد این مدل را نیز در زیر میتوان مشاهده کرد:



شکل 8. ماتریس درهم ریختگی متناظر با مدل آموزش دیده بدون **early stopping** برای داده‌های تست

همانطور که در شکل بالا مشخص است، 669 نمونه معیوب بوده اند و به درستی پیش‌بینی شده‌اند. 217 نمونه معیوب بوده اند اما سالم پیش‌بینی شده‌اند. 235نمونه سالم بوده اما معیوب پیش‌بینی شده‌اند و 9075 نمونه سالم بوده و به درستی سالم پیش‌بینی شده‌اند.

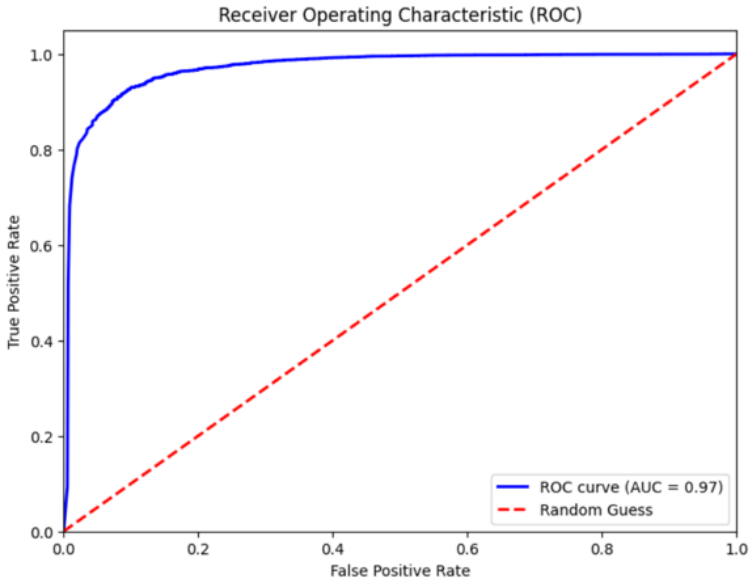
بنابراین، عملکرد مدل را روی هر یک از دو کلاس به صورت زیر میتوان نتیجه گرفت:

	Precision	Recall	F1-Score
Faulty	0.74	0.76	0.75
Healthy	0.98	0.97	0.98

جدول 7. نتایج عملکرد مدل مسئله‌ی طبقه‌بندی بدون **early stopping** روی دو کلاس سالم و معیوب

همچنین، accuracy برای این مدل برابر 96% می‌باشد.

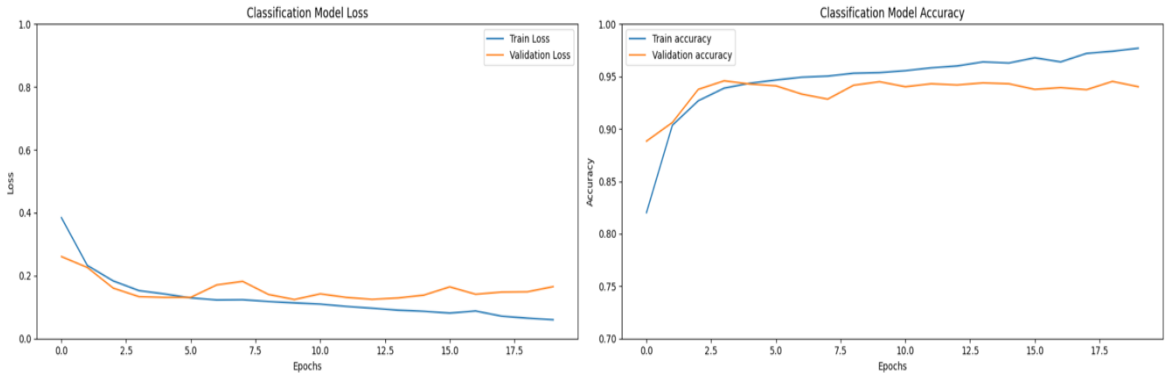
با رسم منحنی ROC برای مدل روی داده‌های تست، شکل زیر بدست می‌اید. از آنجا که منجنی به گوشه‌ی سمت چپ نمودار نزدیک است، میتوان نتیجه گرفت مدل به خوبی عمل می‌کند. همچنین، مساحت نمودار برابر 0.97 می‌باشد که به 1 نزدیک است که بیانگر این است که مدل تا حد خوبی نمونه‌ها را به درستی طبقه بندی می‌کند.



شکل 9. نمودار **ROC**متناظر با مدل مسئله‌ی طبقه‌بندی بدون **early stopping**

در ادامه به بررسی مدل با معماری مشابه با استفاده از رویکرد early stopping می‌پردازیم.

شکل زیر نشان دهده‌ی loss و accuracy خین آموزش این مدل است. همانطور که واضح است، با گذشت 20 ایپاک، دیگر loss داده‌های اعتبارسنجی تغییر چندانی نمی‌کند و آموزش مدل متوقف می‌شود. شکل زیر نشان می‌دهد در طی این ایپاک ها به طول کلی loss در حال کاهش و accuracy در حال افزایش است. در چند ایپاک نهایی اندکی فاصله‌ی بین دو نمودار متناظر با داده‌های آموزشی و اعتبارسنجی افزایش می‌یابد که با توجه به اعداد، به معنی میزان ناچیزی overfittong استو اما به قدری این مقدار کم است که میتوان به راحتی از آن چشم‌پوشی کرد. بنابراین همانطورکه مشخص است، مدل به خوبی در حال یادگیری است.



شکل 10. روند آموزش مدل

در ادامه به ارزیابی مدل می‌پردازیم.

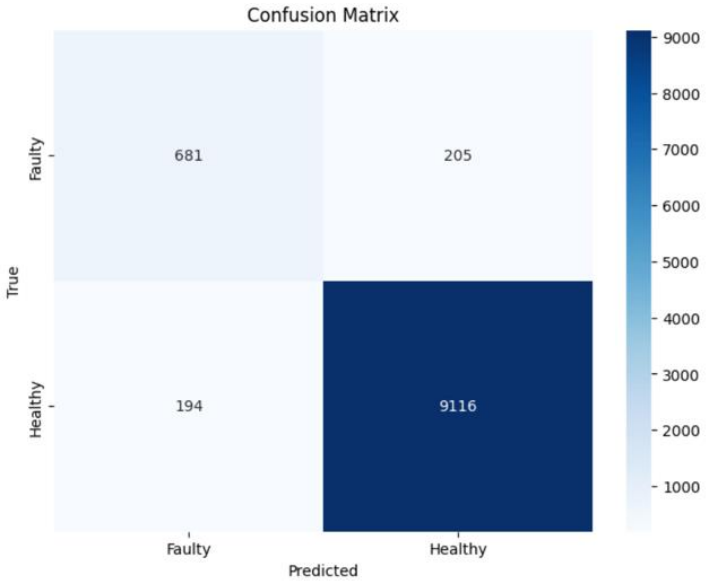
جدول زیر نمایانگر معیارهای مورد ارزیابی روی داده‌های تست هستند:

Accuracy	Precision	Recall	F1-Score
96.1%	97.8%	97.8%	97.8%

جدول 8. نتایج ارزیابی مدل مسئله‌ی طبقه بندی با **early stopping** روی کل داده‌های تست

همانطور که مشخص است، تمام مقادیر جدول فوق نسبت به جدول مشابه در مدل قبلی (جدول ۴)، بیشتر است. بنابراین با استفاده از رویکرد early stopping مدل عملکرد بهتری از خود ارائه می‌دهد و دچار overfitting نمی‌شود.

ماتریس درهم ریختگی برای این مدل به صورت زیر می‌باشد. طبق این شکل می‌توان نتیجه گرفت که 681 نمونه‌ی مغیوب به درستی پیش‌بینس شده‌اند. 205 نمونه‌ی معیوب، به غلط سالم پیش‌بینی شده‌اند. 194نمونه‌ی سالم به غلط معیوب پیش‌بینی شده‌اند و 9116 نمونه‌ی سالم نیز به درستی طلقه بندی شده اند.



شکل 11. ماتریس درهم ریختگی متناظر با مدل آموزش دیده با **early stopping** برای داده‌های تست

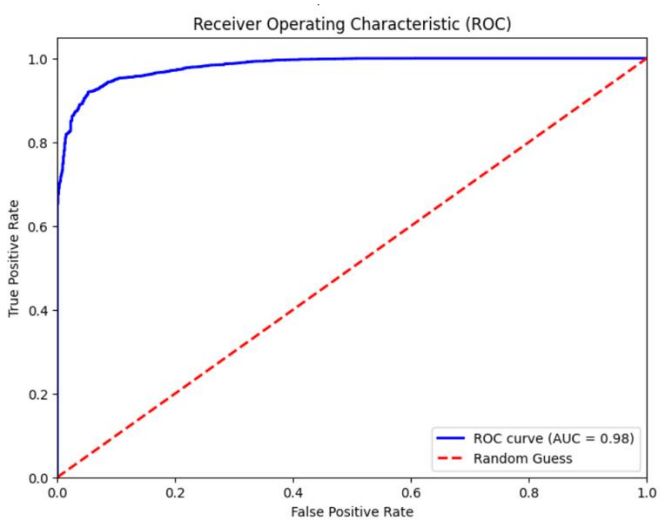
عملکرد این مدل نیز روی هر یک از کلاسها را به تفکیک بررسی میکنیم:

	Precision	Recall	F1-Score
Faulty	0.78	0.77	0.77
Healthy	0.98	0.98	0.98

جدول 9. نتایج عملکرد مدل مسئله‌ی طبقه‌بندی با **early stopping** روی دو کلاس سالم و معیوب

با مقایسه‌ی جدول فوق با جدول مشابه آن برای مدل قبل نتیجه میگیریم به طور کلی مدل با استفاده از رویکرد early stoppingعملکرد بهتری از خود ارائه داده چراکه این رویکر از overfitting چلوگیری می‌کند. همچنین با مقایسه‌ی این جدوب با جدول موجود در مقاله نتیجه میگیریم مدل ما در پیش‌بینی کلاس سالم عملکرد نزدیکی به مدل ارایه شده در مقاله ددارد درحالیکه کلاس معیوب را با دقت کمی کمتر به درستی پیش‌بینی می‌کند. این امر باعث می‌شود که accuracy بدست آمده در مدل ما که برابر 96% است، اندکی از accuracy موجود در مقاله که 98.05% است، کمتر باشد.

در زیر نیز منحنی ROC متناظر با این مدل را مشاهده می‌کنیم که تفاوت چندانی نسبت به مدل قبلی ندارد حز اینکه اندکی در ابتدای مسیر، شیب نمودار بیشتر است. این تغییر اندکی منجر شده که مساحت زیر نمودار در این حالت اندکی بیشتر از قبل و برابر با 0.98 شود که باز هم به معنی این است که classifierخوبی را اموزش داده‌ایم. ضمن اینکه نسبت به مدل قبل، با وجود معماری یکسان شبکه، اندکی دقت آن افزایش یافته است و عمل طبقه‌بندی را بهتر انجام می‌دهد.



شکل 12. نمودار **ROC**متناظر با مدل مسئله‌ی طبقه‌بندی با رویکرد **early stopping**

با مقایسه‌ی دو مدل فوق میتوان نتیجه گرفت که استفاده از رویکرد early stopping از رخداد overfitting جلوگیری می‌کند و منجر می‌شود classifier ما عملکرد بهتری از خود ارائه دهد.

Regression:

در این بخش قصد داریم همانطور که در صورت پروژه گفته شده، دو مدل بدون رویکرد early stopping و با این رویکرد را برای مسئله‌ی regression بررسی و با هم مقایسه کنیم.

برای هر دو مدل در این بخش نیز پارامترها را به صورت زیر در نظر میگیریم:

- فعال‌ساز adam
- تابع هزینه‌ی mse
- تعداد ایپاک 100
- Batch size برابر 200

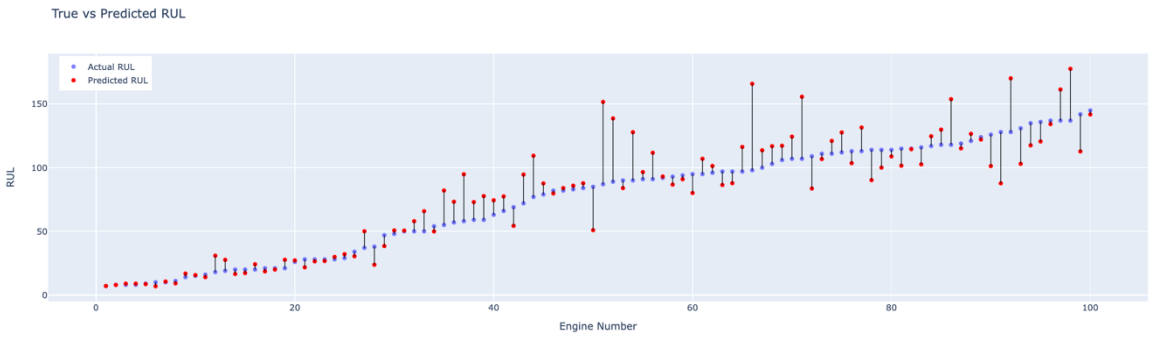
ابتدا حالت بدون early stopping را بررسی می‌کنیم.

برای این مدل، مقیارهای MAPE، MAE، MSE و RMSE به صورت زیر به دست آمده‌اند:

MAPE	MAE	MSE	RMSE
156.718	13.280	374.018	19.340

جدول 10. نتایج ارزیابی مدل مسئله‌ی **regression** بدون **early stopping** روی آخرین پنجره متناظر با هر موتور

نتایج پیش‌بینی مدل روی داده‌های تست به صورت مقابل است. لازمه به ذکر است که برای وضوح بیشتر، مقادیر به صورت صعودی مرتب شده اند.



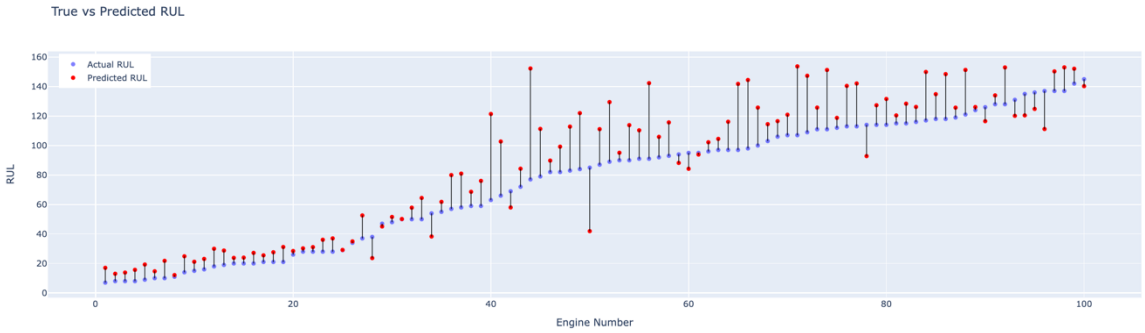
شکل 13. نتایج پیش‌بینی **RUL** توسط مدل **CNN-LSTM** بر روی داده‌های تست بدون **early stopping**

برای حالتی که early stopping را در نظر بگیریم نیز نتایج به صورت زیر است:

MAPE	MAE	MSE	RMSE
155.545	36.029	1954.732	44.212

جدول 11. . نتایج ارزیابی مدل مسئله‌ی **regression** با **early stopping** روی آخرین پنجره متناظر با هر موتور

در شکل زیر نیز میتوان نتایج پیش‌بینی مدل را بررسی کرد:



شکل 14. نتایج پیش‌بینی مدل

طبق نتایجی که در بررسی ای دو حالت گرفتیم، علی رغم اینکه انتظار داشتیم با استفاده از early stopping احتمال رخداد overfitting کاهش پیدا کند و مدل با خطای کمتری در پیش‌بینی مواجه شود و عملکرد بهتری از خود ارائه دهد، اما ما در صورتی که تمام ایپاک‌ها اجرا شدند، با خطای کمتری مواجه شدیم. این ممکن است بخاطر انتخاب patience باشد و اگر مقدار آن را بزرگتر انتخاب می‌کردیم، ممکن بود نتایج بهتری بدست می‌اوردیم. در واقع، احتمالا با این انتخاب، از تمام پتانسبل مدل برای آموزش استفاده نکرده‌ایم.

حال نوبت آن رسیده که عملکرد مدل را روی تمام پنجره‌ها بررسی کنیم و نه تنها آخرین پنجره‌ی متناظر با هر موتور. نتایج زیر، مربوط به حالتی است که early stopping اعمال نشده.

MAPE	MAE	MSE	RMSE
67.552	29.608	1618.245	40.227

جدول 12. . نتایج ارزیابی مدل مسئله‌ی **regression** بدون **early stopping** روی تمام پنجره‌ها

با استفاده از early stopping نیز نتایج به صورت زیر است:

MAPE	MAE	MSE	RMSE
60.141	27.659	1596.937	39.962

جدول 13. نتایج ارزیابی مدل مسئله‌ی **regression** با **early stopping** روی تمام پنجره‌ها

همانطور که انتظار می‌رفت، در این حالت که تمام پنجره‌ها در نظر گرفته می‌شود،گویی مقدار patience انتخاب شده مناسب است و با اعمال early stopping، خطاهای پیش‌بینی مدل کاهش می‌یابد.

نتایجی که ما در این پژوهش برای مسئله‌ی regression گرفتیم به ترتیب از بالا به پایین خطای پیش‌بینی‌شان افزایش می‌یابد:

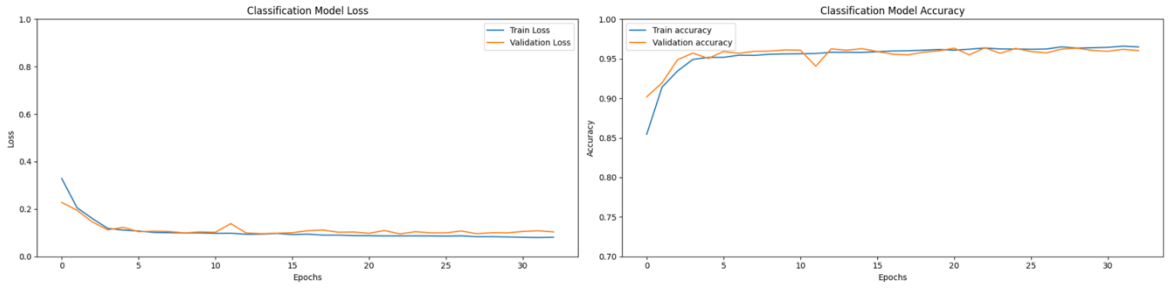
1. در نظر گرفتن آخرین پنجره متناظر با هر موتور بدون اعمال early stopping
2. در نظر گرفتن تمام پنجره‌های متناظر با هر موتور با اعمال early stopping
3. در نظر گرفتن تمام پنجره‌های متناظر با هر موتور بدون اعمال early stopping
4. در نظر گرفتن آخرین پنجره متناظر با هر موتور بدون اعمال early stopping

این نتایج درحالیست که انتظار داشتیم با اعمال early stopping به نتایج بهتری دست پیدا کنیم که علت آن دربالاتر توضیح داده شد.

۲-۳. مقایسه با مدل‌های پایه

ابتدا هر دو مسئله‌ی classification و regression را با مدل پایه‌ی CNN بررسی می‌کنیم. این مدل، صرفا شمانل بخش کانولوشنی مدلی است مه تا به حال استفاده میکردیم و تعداد کل پارامترهای آن برابر 9441 عدد می‌باشد!

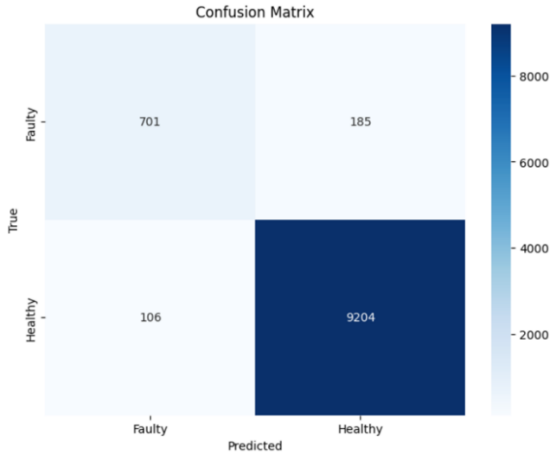
با آموزش این مدل با همان پارامترهای قبلی، نتایج زیر بدست می‌اید:



شکل 15. تغییرات **loss** و **accuracy** برای مدل **CNN**

Accuracy	Precision	Recall	F1-Score
97%	98%	98%	98%

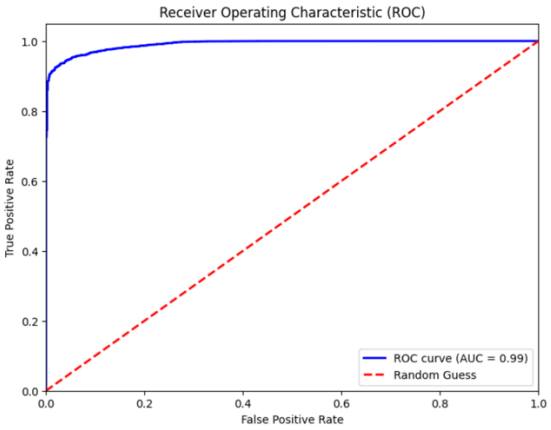
جدول 14. نتایج ارزیابی مدل **CNN** رو داده‌های تست



شکل 16. ماتریس در هم ریختگی متناظر با مدل **CNN** برای مسئله‌ی طبقه بندی

	Precision	Recall	F1-Score
Faulty	0.78	0.77	0.77
Healthy	0.98	0.98	0.98

جدول 15. نتایج عملکرد مدل **CNN** مسئله‌ی طبقه‌بندی با **early stopping** روی دو کلاس سالم و معیوب



شکل 17. منحنی **ROC** متناظر با مدل **CNN** با **early stopping**

accuracy نیز در این حالت برابر ۹۷ درصد می‌باشد

حال، مسئله‌ی regression را با این مدل بررسی می‌کنیم. نتایج آن به شکل زیر است:

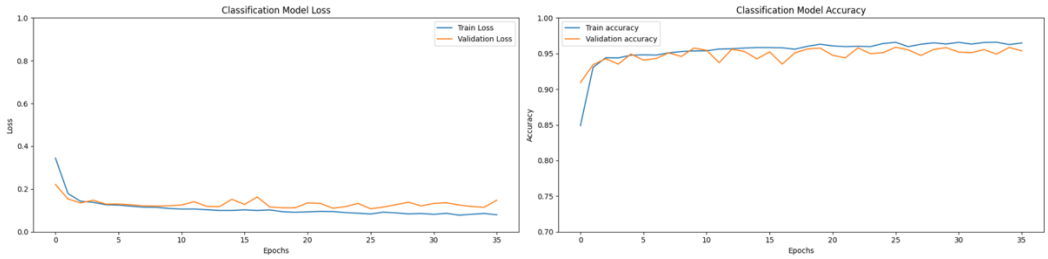
MAPE	MAE	MSE	RMSE
100.078	75.550	7434.595	84.224

جدول 16. نتایج ارزیابی مدل **CNN** مسئله‌ی **regression** با **early stopping** روی آخرین پنجره متناظر با هر موتور

با بررسی دو مدل فوق و حالات قبلی میتوان نتیجه گرفت برای حل مسئله‌ی regression، مدل ارائه شده در مقاله عملکرد بسیار بهتری دارد. اما برای classification ما به نتایج چندان متفاوتی نرسیدیم.

در نهایت، نوبت به بررسی مدل پایه‌ی LSTM می‌رسد.

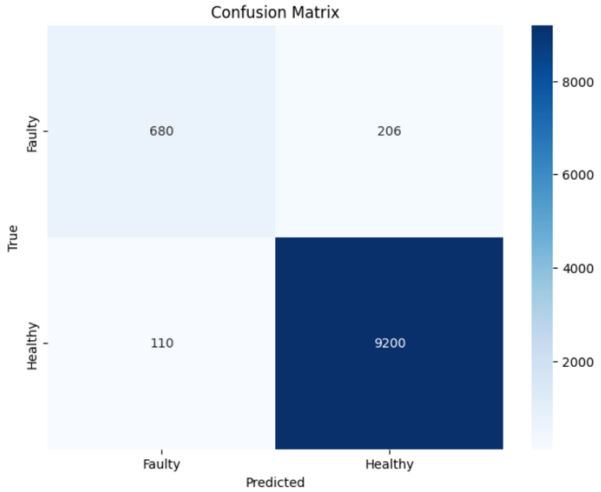
برای مسئله‌ی طبقه بندی با استفاده از این مدل، نتایج زیر بدست آمده است:



شکل 18. تغییرات **loss** و **accuracy** برای مدل **LSTM**

نتایج متریکهای مورد بررسی در این حالت به شکل زیر است:

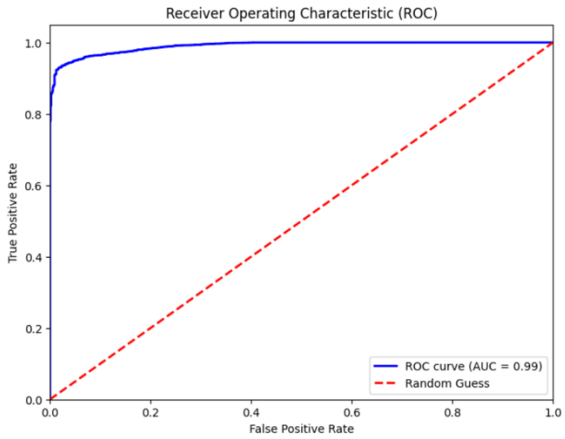
Accuracy	Precision	Recall	F1-Score
97%	98%	99%	98%



شکل**19** . ماتریس در هم ریختگی متناظر با مدل **LSTM** برای مسئله‌ی طبقه بندی با **early stopping**

	Precision	Recall	F1-Score
Faulty	0.86	0.77	0.81
Healthy	0.98	0.99	0.98

جدول 17. نتایج عملکرد مدل **LSTM** مسئله‌ی طبقه‌بندی با **early stopping** روی دو کلاس سالم و معیوب

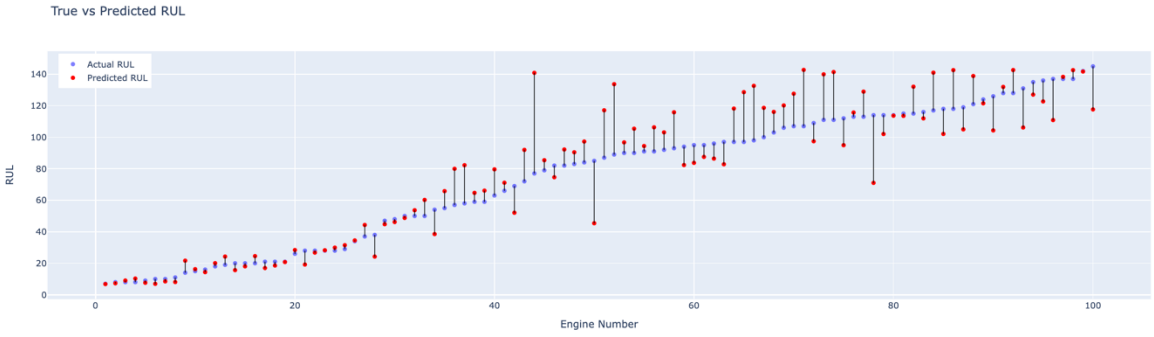


شکل 20. منحنی **ROC** متناظر با مدل **LSTM** با **early stopping**

در نهایت نیز مدل LSTM را برای مسئله‌یregression بررسی میکنیم:

MAPE	MAE	MSE	RMSE
155.077	12.233	287.516	16.956

جدول 18. نتایج ارزیابی مدل **LSTM** مسئله‌ی **regression** با **early stopping** روی آخرین پنجره متناظر با هر موتور



شکل 21. نتایج پیش‌بینی **RUL** توسط مدل **LSTM** بر روی داده‌های تست با **early stopping**

برای مقایسه دوم مدل پایه‌ی CNN و LSTM میتوان گفت که تقوا چندانی در مواجه با مسئله‌ی طبقه‌بندی که در حالت حل آن هستیم، ندارند. اما در مورد مسئله‌یregression ، مدل پایه‌ی LSTM عملکرد بسیار بهتری دارد.

در مورد مقایسه‌ی کلی مدل‌های پایه با مدل پیشنهادی مقاله نیز ما به نتایج متفاوتی رسیدیم. مدل LSTM با رویکرد early stopping بهتری نتایج را له ما داد. پس از آن، مدل پیشنهادی مقاله در رتبه‌ی دوم قرار میگیرد و نهایتا مدل پایه‌ی CNN.

در مورد مسئله‌ی طبقه بندی برای تمام مدل‌ها تقریبا تمام accuracy ها نزدیک و مشابه بودند، چه رویکرد early stoppong را استفاده کردیم، چه نه. اما در مورد مسئله‌ی regression، عملکرد ها متفوت بود و در مجموع، درم واردی خطای زیادی در پیش‌بینی داشتیم که در تصاویر قابل مشاهده است.

بنابراین، همانطور که گفته شد، در حالتی که از early stopping بهره گرفتیم و از مدل پایه‌ی LSTM استفاده کردیم، بهترین نتایج کسب شده، کما اینکه این نتایج با آنچه در مقاله ذکر شده متفاوت است.