



به نام خدا

دانشگاه تهران



دانشکده مهندسی برق و کامپیوتر

## درس شبکه‌های عصبی و یادگیری عمیق

### تمرین چهارم

علی خرم فر	نام و نام خانوادگی	پرسش ۱ و ۲
khoramfar@ut.ac.ir	رایانامه	
علی رمضانی	نام و نام خانوادگی	پرسش ۱ و ۲
Ali.ramezani.۹۶@ut.ac.ir	رایانامه	
۱۴۰۳/۰۳/۰۹	تاریخ ارسال پاسخ	

- هر دو نویسنده در هر دو پرسش همکاری داشته اند

## • فهرست

۱	پرسش ۱ - تحلیل احساسات متن فارسی
۱	۱. مجموعه داده
۲	توزيع کلاس‌های ستون احساسات
۳	۲-۱. پیش‌پردازش داده‌ها
۵	اعمال پیش‌پردازش‌های پیاده‌سازی شده
۶	کتابخانه‌های استفاده شده برای پیش‌پردازش
۷	۳-۳. نمایش ویژگی
۸	استفاده از توکن‌ساز ParsBERT
۸	بردار تعییه یا Embedding
۹	تکنیک‌های مدیریت حافظه
۹	کتابخانه gc برای Garbage Collection
۱۱	تقسیم داده‌ها
۱۲	معماری LSTM
۱۳	مدل CNN-LSTM: توضیح و پیاده‌سازی با توجه به مقاله
۱۴	جستجوی حریصانه پارامترهای بهینه
۱۶	مدل CNN ساده
۱۸	مدل LSTM ساده
۱۹	بررسی نقاط ضعف و قوت مدل‌ها
۲۰	ادغام هردو مدل
۲۰	توییت‌های سیاسی در مقابل توییت‌های عمومی
۲۱	۱-۵. ارزیابی
	فهرست تصاویر
	ت

۲۴	مقایسه روش های میانگین گیری
۲۵	۱-۶. امتیازی - کیسه کلمات
۲۶	تحلیل نتایج
۲۷	پرسش ۲ - سامانه های سایر فیزیکی: نگهداری هوشمند
۲۷	بخش ۲-۱: پیش پردازش داده ها:
۳۳	بخش ۲-۲: مدل سازی و ارزیابی:
۴۶	بخش ۲-۳: مقایسه با مدل های پایه:

## فهرست تصاویر

۱.....	شکل ۱ بارگذاری داده‌ها
۲.....	شکل ۲ تبدیل داده‌ها به ۲ ستون
۲.....	شکل ۳ تعداد نمونه‌های هر کلاس
۵.....	شکل ۴ اعمال پیش‌پردازش‌های لازم
۵.....	شکل ۵ اعمال تمامی پیش‌پردازش‌ها - کاهش دقت مدل
۶.....	شکل ۶ اعمال پیش‌پردازش‌ها به جز stemming و normalization - افزایش دقت مدل
۶.....	شکل ۷ اعمال پیش‌پردازش بر روی دیتاست
۸.....	شکل ۸ اعمال توکن‌ساز PARSBERT
۹.....	شکل ۹ استخراج تعابیه‌ها به صورت دسته‌ای
۱۰.....	شکل ۱۰ فضای بردار کلمات - تبدیل شده به ۲ بعد
۱۱.....	شکل ۱۱ تقسیم داده‌ها به مجموعه‌های آموزشی، اعتبارسنجی و تست
۱۱.....	شکل ۱۲ ساخت دیتاست‌های آزمون، اعتبارسنجی و تست
۱۲.....	شکل ۱۳ معماری LSTM
۱۳.....	شکل ۱۴ معماری مدل CNN-LSTM
۱۴.....	شکل ۱۵ فضای جستجوی حریصانه
۱۴.....	شکل ۱۶ نتیجه آموزش مدل برای حالت اول
۱۵.....	شکل ۱۷ نمودار خطای دقت نسبت به epoch برای حالت اول - دقت اعتبارسنجی حدود ۷۸ درصد
۱۵.....	شکل ۱۸ نتیجه آموزش بر اساس بهترین پارامترها
۱۶.....	شکل ۱۹ نمودار خطای دقت نسبت به epoch برای بهترین حالت - دقت اعتبارسنجی حدود ۷۹ درصد
۱۷.....	شکل ۲۰ معماری مدل CNN ساده
۱۷.....	شکل ۲۱ نمودار خطای دقت نسبت به epoch برای مدل CNN ساده
۱۸.....	شکل ۲۲ معماری مدل LSTM ساده
۱۸.....	شکل ۲۳ نمودار خطای دقت نسبت به epoch برای مدل LSTM ساده
۲۲.....	شکل ۲۴ ماتریس آشفتگی مدل CNN-LSTM
۲۲.....	شکل ۲۵ ماتریس آشفتگی مدل CNN
۲۳.....	شکل ۲۶ ماتریس آشفتگی مدل LSTM
۲۵.....	شکل ۲۷ روش کیسه کلمات

شکل ۲۸ - داده های آموزشی	۲۷
شکل ۲۹ - RUL داده های تست	۲۸
شکل ۳۰ - حذف ستون های بی استفاده در تخمین عمر	۲۸
شکل ۳۱ - محدودیت حد بالای عمر به ۱۳۰ سایکل	۲۹
شکل ۳۲ - طول پنجره برای طبقه بندی	۲۹
شکل ۳۳ - کد بخش لیبل زدن	۲۹
شکل ۳۴ - ستون های اضافه شده بخش لیبل زدن	۳۰
شکل ۳۵ - تعداد دیتای هر کلاس	۳۰
شکل ۳۶ - استفاده از داده های RUL در بخش تست	۳۰
شکل ۳۷ - تعداد داده های هر کلاس در دیتای تست	۳۱
شکل ۳۸ - جداسازی و نرمال سازی ویژگی های تست و آموزش	۳۱
شکل ۳۹ - پنجره زمانی پیشنهادی مقاله	۳۱
شکل ۴۰ - استراید پیشنهادی مقاله	۳۲
شکل ۴۱ - کد ایجاد پنجره های زمانی	۳۲
شکل ۴۲ - shape بردارهای داده، و لیبل ها	۳۲
شکل ۴۳ - تعداد داده های هر کلاس در داده آموزش	۳۳
شکل ۴۴ - داده و لیبل ها در داده های تست	۳۳
شکل ۴۵ - تعداد داده در هر کلاس داده تست	۳۳
شکل ۴۶ - ساخت دیتاست مورد نیاز از روی داده ها	۳۳
شکل ۴۷ - معماری مدل CNN – LSTM	۳۴
شکل ۴۸ - عدم ذکر لایه Fully Connected	۳۴
شکل ۴۹ - اشاره به Fully Connected نهایی	۳۴
شکل ۵۰ - ساختار مدل CNN-LSTM CLS	۳۵
شکل ۵۱ - نمودار آموزش مدل CNN - LSTM CLs	۳۵
شکل ۵۲ - نتایج مدل CNN - LSTM CLs	۳۶
شکل ۵۳ - ماتریس درهم ریختگی مدل CNN - LSTM CLs	۳۶
شکل ۵۴ - مدل ROC - CNN-LSTM CLs	۳۷
شکل ۵۵ - آموزش CNN – LSTM CLs با Early Stopping	۳۷
شکل ۵۶ - نمودار های آموزش مدل با Early stoping	۳۸

۳۸	..... شکل ۵۷ - معیار های ارزیابی دقت مدل با Early Stopping
۳۹	..... شکل ۵۸ - ماتریس درهم ریختگی با Early Stopping
۳۹	..... شکل ۵۹ ROC - مدل با Early Stopping
۴۰	..... شکل ۶۰ - کد ساختار مدل
۴۰	..... شکل ۶۱ - سخت تر بودن شبکه Classification Regressive نسبت به
۴۰	..... شکل ۶۲ - ساختار شبکه Regressor
۴۱	..... شکل ۶۳ - نمودارهای آموزش شبکه در هر ۴ معیار خواسته شده
۴۱	..... شکل ۶۴ - معیار ها روی همه پنجره های داده تست
۴۱	..... شکل ۶۵ - معیار ها روی آخرین پنجره زمانی هر داده تست
۴۲	..... شکل ۶۶ - قدرت مدل در پیش بینی مقدار RUL هر داده
۴۲	..... شکل ۶۷ - تصویر از مقاله برای مقایسه با نتایج ما
۴۳	..... شکل ۶۸ - در مواد مکانیکال Fatigue
۴۴	..... شکل ۶۹ - نمودارهای آموزش مدل early stopping با Regressor
۴۵	..... شکل ۷۰ - مقایسه مقادیر واقعی و تخمینی در مدل
۴۶	..... شکل ۷۱ - معماری CNN - Cls
۴۷	..... شکل ۷۲ - ساختار مدل CNN - Cls
۴۷	..... شکل ۷۳ - نمودار های ACC و Loss روی CNN - CLs
۴۸	..... شکل ۷۴ - نتایج مدل CLs CNN
۴۸	..... شکل ۷۵ - ماتریس پراکندگی مدل CNN - Cls
۴۹	..... شکل ۷۷ - منحنی CNN CLs - ROC
۵۰	..... شکل ۷۸ - مدل CNN - Regressor
۵۰	..... شکل ۷۹ - ساختار مدل CNN - Regressor
۵۱	..... شکل ۸۰ - نمودار های ارزیابی CNN - Regressor
۵۱	..... شکل ۸۱ - نتایج مدل CNN Regressor
۵۲	..... شکل ۸۲ - مقایسه دیتای واقعی و پیش بینی توسط مدل CNN - Regressor
۵۳	..... شکل ۸۴ - ساختار مدل LSTM - CLs
۵۳	..... شکل ۸۵ - ساختار مدل CNN - Cls
۵۴	..... شکل ۸۶ - نمودار های ACC و Loss روی LSTM - CLs
۵۴	..... شکل ۸۷ - نتایج مدل CLs CNN

۵۵	..... شکل ۸۸ - ماتریس پراکندگی مدل LSTM – Cls
۵۶	..... شکل ۹۰ - منحنی LSTM CLs - ROC
۵۷	..... شکل ۹۱ - مدل LSTM - Regressor
۵۷	..... شکل ۹۲ - ساختار مدل LSTM - Regressor
۵۸	..... شکل ۹۳ - نمودار های ارزیابی LSTM - Regressor
۵۸	..... شکل ۹۴ - نتایج مدل LSTM Regressor
۵۹	..... شکل ۹۵ - مقایسه دیتای واقعی و پیش بینی توسط مدل LSTM - Regressor

## فهرست جداول:

۲۴	جدول ۱ خروجی روش‌های یادگیری عمیق و یادگیری ماشین
۴۶	جدول ۲ - مقایسه مدل CNN-LSTM Classifier
۴۶	جدول ۳ - مقایسه مدل‌های CNN-LSTM Regressor
۴۹	جدول ۴ - مقایسه مدل CNN CLs , CNN-LSTM CLs
۵۲	جدول ۵ - مقایسه مدل CNN CLs , CNN-LSTM CLs
۵۶	جدول ۶ - مقایسه مدل LSTM CLs , CNN-LSTM CLs
۵۹	جدول ۷ - مقایسه مدل CNN CLs , CNN-LSTM CLs

## پرسش ۱ - تحلیل احساسات متن فارسی

در این مسئله قصد داریم که دیتاست ارائه شده از توییت‌های فارسی را بر اساس ۶ حالت مختلف دسته‌بندی کنیم. در این مسئله از روش‌های مختلف یادگیری عمیق و یادگیری ماشین استفاده خواهیم کرد و در انتها آن‌ها را با هم مقایسه می‌کنیم.

### ۱-۱. مجموعه داده

ابتدا کتابخانه‌های لازم را نصب می‌کنیم و سپس داده‌ها را بارگذاری می‌کنیم. داده‌ها در این مجموعه بر حسب هر احساس در فایل جداگانه قرار دارند. داده‌های مربوط به هر احساس را از فایل‌های CSV مربوطه بارگذاری می‌کنیم و چند سطر اول آن‌ها را نمایش می‌دهیم:

Load Dataset							
+ Code		+ Markdown					
<pre>anger = pd.read_csv('kaggle/input/persian-tweets-emotional-dataset/anger.csv') disgust = pd.read_csv('kaggle/input/persian-tweets-emotional-dataset/disgust.csv') fear = pd.read_csv('kaggle/input/persian-tweets-emotional-dataset/fear.csv') joy = pd.read_csv('kaggle/input/persian-tweets-emotional-dataset/joy.csv') sad = pd.read_csv('kaggle/input/persian-tweets-emotional-dataset/sad.csv') surprise = pd.read_csv('kaggle/input/persian-tweets-emotional-dataset/surprise.csv')  tweets = pd.concat([anger, disgust, fear, joy, sad, surprise], ignore_index=True) tweets.head()</pre>							
tweet	replyCount	retweetCount	likeCount	quoteCount	hashtags	sourceLabel	emotion
دیشب خواب دیدم بمی چیزی زدن نورش خبلی خیره کن	0	3	2	0	['No2IR']	Twitter Web App	anger
رجان‌آن‌تر زدی بر ریشه‌آم، جوانه رو بید جای زخم	0	0	8	0	['سن_کاف']	Twitter for Android	anger
سپدر سوخته‌ای که بایام بهم میگه دو معنی داره که	1	0	11	0	['بدر_ابرانی']	Twitter for Android	anger
با خود مواجه شوید و احتم نکنید. اقتدار در نگاه	0	0	1	0	['جذبه', 'اخ']	Twitter for iPhone	anger
با این همه آمده تو را در شادی و در عالم نوشتند	4	6	36	0	['بن_اعظیم']	Twitter Web App	anger

شکل ۱ بارگذاری داده‌ها

فایل‌ها را با هم ادغام کرده و دیتاست اصلی را می‌سازیم که شامل ۱۱۳۸۲۹ سطر و ۸ ستون است. هر سطر نمایانگر یک توییت و ستون‌های مختلف شامل اطلاعاتی درباره هر توییت هستند. همانطور که در شکل بالا مشاهده می‌شود این دیتاست شامل ستون‌های مختلفی است. هدف اصلی این پرسش به نحوی است که ما فقط از دو ستون متن توییت و emotion یا احساس هر توییت که لیبل آن است، استفاده می‌کنیم.

لیست ستون‌های دیتاست اصلی به شرح زیر است:

'tweet', 'replyCount', 'retweetCount', 'likeCount', 'quoteCount', 'hashtags', 'sourceLabel', 'emotion'

در نهایت فقط ۲ ستون آن را انتخاب کرده و دیتاست مورد نظر این پرسش را می‌سازیم:

### Dataset from 'tweet' and 'emotion' columns:

```
data = tweets[['tweet', 'emotion']].copy()  
print(data.shape)  
data.head()
```

(113829, 2)

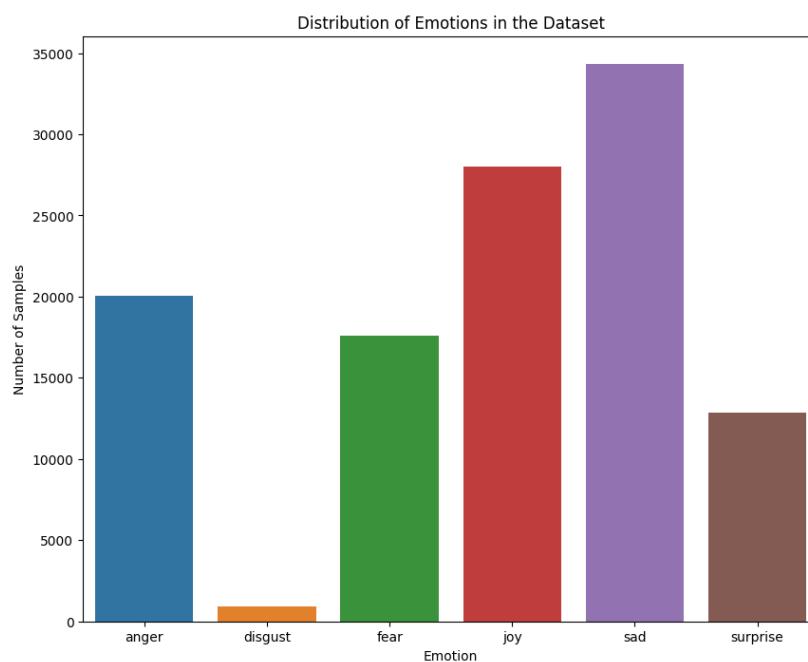
	tweet	emotion
0	دیشب خواب دیدم بمی چیری زدن نورش خیلی خیره کن	anger
1	... ران\نتر زدی بر ریشه‌آم، حوانه رو بید جای رخم	anger
2	... پدر سوخته ای که بایام بهم میگه دو معنی داره که	anger
3	... با خود مواجه شوید و اخم نکنید. اقتدار در نگاه	anger
4	... با این همه \n\مدح تو را در شادی و در غم نوشتند	anger

شکل ۲ تبدیل داده‌ها به ۲ ستون

### توزیع کلاس‌های ستون احساسات

ابتدا کلاس‌های موجود در ستون emotion را استخراج کرده و سپس با استفاده از کتابخانه matplotlib

نمودار میله‌ای را رسم می‌کنیم:



شکل ۳ تعداد نمونه‌های هر کلاس

که کلاس‌ها به شرح زیر هستند:

[anger' 'disgust' 'fear' 'joy' 'sad' 'surprise']

همانطور که در نمودار بالا مشاهده می‌کنیم، تعداد نمونه‌های هر کلاس به به صورت زیر است:

غم sad: ۳۴۳۲۸

شادی joy: ۲۸۰۲۴

خشم anger: ۲۰۰۶۹

ترس fear: ۱۷۶۲۴

تعجب surprise: ۱۲۸۵۹

تنفر disgust: ۹۲۵

این نمودار به خوبی نشان می‌دهد که تعداد نمونه‌های کلاس‌های مختلف در مجموعه داده متفاوت است و کلاس‌هایی مانند غم و شادی دارای تعداد تويیت‌های بیشتری نسبت به سایر کلاس‌ها هستند. کلاس غم بیشترین تعداد نمونه و کلاس تنفر کمترین تعداد نمونه را در این مجموعه داده دارد. این موضوع برای متوازن بودن داده‌ها خصوصاً در مرحله ارزیابی حائز اهمیت است.

## ۱-۲. پیش‌پردازش داده‌ها

در این بخش، مراحل پیش‌پردازش ذکر شده در مقاله را اعمال می‌کنیم. مراحل پیش‌پردازش متن شامل حذف تگ‌های HTML و URL‌ها، حذف کاراکترهای تکراری، حذف واژگان stopwords، جایگزینی ایموجی‌ها با معادل متنی، اعمال ریشه‌یابی يا stemming و تصحیح املای نادرست پیاده‌سازی شد. هر مرحله با یک مثال از قبل و بعد از پیش‌پردازش را در زیر مشاهده می‌کنیم:

### حذف تگ‌های HTML و URL‌ها:

تگ‌های HTML و URL‌ها معمولاً در متن تويیت‌هایی که استخراج شده‌اند وجود دارند اما برای تحلیل متن مفید نیستند.

Before: لینک زیر یک لینک تست است: [https://test.com <html> test </html>](https://test.com)

After: لینک زیر یک لینک تست است: test

### حذف کاراکترهای تکراری:

کاراکترهای تکراری مانند خوووووب می‌توانند نویز ایجاد کنند پس با حذف آن‌ها متن یکپارچه‌تری خواهیم داشت:

Before: امرووووز هوا خیلییی خوبهههههه

After: امروز هوا خیلی خوبه

### حذف واژگان غیرمفید یا ایست واژه (stopwords):

واژگان غیرمفید مانند و، یا، است اطلاعات کمی به تحلیل اضافه می‌کند و باید حذف شوند. همچنین این مورد برای روش کیسه کلمات مهم است زیرا که در صورتی که آن‌ها را حذف نکنیم تکرار بیهوده در مدل محسوب می‌شوند.

Before: این یک مثال برای حذف ایست واژه است

After: مثال حذف ایست واژه

### جایگزینی ایموجی با معادل متنی

اموجی‌ها می‌توانند معانی مهمی در متن داشته باشند و نقش آن‌ها خصوصاً در تحلیل احساسات بسیار مهم است. جایگزینی آن‌ها با معادل‌های متنی می‌تواند به تحلیل بهتر مدل کمک کند.

Before: ٩٣ چقدر خوبه هوا 😊

After: چقدر خوبه هوا:smiling\_face\_with\_heart-eyes::smiling\_face\_with\_smiling\_eyes:

### ریشه‌یابی (stemming)

ریشه‌یابی کلمات و تبدیل آن‌ها به شکل پایه‌ی کلمه کمک می‌کند تا تنوع کلمات کاهش یابد و تحلیل احساسات ساده‌تر شود.

Before: کتاب‌ها

After: کتاب

### اصلاح املای نادرست کلمات (Normalizer)

منبع تحلیل انجام شده در این مقاله و دیتاست ارائه شده توییت‌هایی هستند که ممکن است کاربر سریع نوشته و یا دقت کافی نداشته باشد و شامل غلط املایی باشند.

### اعمال پیش‌پردازش‌های پیاده‌سازی شده

باتوجه به اینکه ممکن است برخی از پیش‌پردازش‌هایی که بررسی شدند، تاثیر مثبتی نداشته باشند، نیازی نیست که تمامی آن‌ها را اعمال کنیم. ابتدا یکتابع نوشته که بتوانیم کنترل بهتری بر روی مواردی که نیاز است اعمال شوند و ترتیب آن‌ها داشته باشیم.

```
def preprocess_text(text):
    text = remove_html_urls(text)
    text = remove_repetitive_chars(text)
    text = replace_emojis(text)
    text = correct_spelling(text)
    text = remove_stopwords(text)
    # text = stem_text(text)
    return text

# Example
example_text = "لینک های ناشناس خطرناک هستند" لینک های ناشناس خطرناک هستند https://test.com 😊"
print("Before:", example_text)
print("After:", preprocess_text(example_text))

Before: لینک های ناشناس خطرناک هستند https://test.com 😊
After: : لینک‌های ناشناس خطرناک : smiling_face_with_smiling_eyes :
```

شکل ۴ اعمال پیش‌پردازش‌های لازم

در ابتدا تمامی پیش‌پردازش‌های مذکور اعمال شد ولی بعدا متوجه شدیم که اگر stemming را انجام ندهیم دقت مدل افزایش پیدا می‌کند. پس از اعمال آن صرف نظر کردیم. این مورد در یکی از مراحل جستجوی حریصانه در شکل زیر مشخص است:

اعمال تمامی پیش‌پردازش‌ها:

```
Training with batch_size=8, learning_rate=0.001, optimizer=Adam
Epoch 1/10, Train Loss: 1.1361, Train Accuracy: 0.5656, Val Loss: 0.8894, Val Accuracy: 0.6769
Epoch 2/10, Train Loss: 0.8356, Train Accuracy: 0.6996, Val Loss: 0.8060, Val Accuracy: 0.7085
Epoch 3/10, Train Loss: 0.7515, Train Accuracy: 0.7314, Val Loss: 0.7510, Val Accuracy: 0.7307
Epoch 4/10, Train Loss: 0.7007, Train Accuracy: 0.7484, Val Loss: 0.7499, Val Accuracy: 0.7287
Epoch 5/10, Train Loss: 0.6619, Train Accuracy: 0.7636, Val Loss: 0.7487, Val Accuracy: 0.7356
Epoch 6/10, Train Loss: 0.6349, Train Accuracy: 0.7724, Val Loss: 0.7402, Val Accuracy: 0.7382
Epoch 7/10, Train Loss: 0.6134, Train Accuracy: 0.7810, Val Loss: 0.7594, Val Accuracy: 0.7348
Epoch 8/10, Train Loss: 0.5954, Train Accuracy: 0.7865, Val Loss: 0.7551, Val Accuracy: 0.7336
Epoch 9/10, Train Loss: 0.5812, Train Accuracy: 0.7911, Val Loss: 0.7562, Val Accuracy: 0.7404
Epoch 10/10, Train Loss: 0.5664, Train Accuracy: 0.7960, Val Loss: 0.7617, Val Accuracy: 0.739
```

شکل ۵ اعمال تمامی پیش‌پردازش‌ها - کاهش دقت مدل

در آزمایش بعدی، stemming و normalization حذف شد و نتیجه زیر حاصل شد:

```

Training with batch_size=8, learning_rate=0.001, optimizer=Adam
Epoch 1/10, Train Loss: 1.0490, Train Accuracy: 0.6002, Val Loss: 0.7591, Val Accuracy: 0.7258
Epoch 2/10, Train Loss: 0.6888, Train Accuracy: 0.7533, Val Loss: 0.6524, Val Accuracy: 0.7680
Epoch 3/10, Train Loss: 0.6123, Train Accuracy: 0.7817, Val Loss: 0.6311, Val Accuracy: 0.7750
Epoch 4/10, Train Loss: 0.5744, Train Accuracy: 0.7958, Val Loss: 0.6052, Val Accuracy: 0.7828
Epoch 5/10, Train Loss: 0.5398, Train Accuracy: 0.8070, Val Loss: 0.6097, Val Accuracy: 0.7849
Epoch 6/10, Train Loss: 0.5191, Train Accuracy: 0.8147, Val Loss: 0.6220, Val Accuracy: 0.7840
Epoch 7/10, Train Loss: 0.4993, Train Accuracy: 0.8229, Val Loss: 0.6258, Val Accuracy: 0.7836
Epoch 8/10, Train Loss: 0.4844, Train Accuracy: 0.8266, Val Loss: 0.6226, Val Accuracy: 0.7854
Epoch 9/10, Train Loss: 0.4719, Train Accuracy: 0.8322, Val Loss: 0.6108, Val Accuracy: 0.7878
Epoch 10/10, Train Loss: 0.4588, Train Accuracy: 0.8366, Val Loss: 0.6309, Val Accuracy: 0.7880

```

## شکل ۶ اعمال پیش‌پردازش‌ها به جزNormalization و stemming دقت مدل

در نهایت تصمیم گرفته شد فقط از stemming صرف نظر شود زیرا که به دقت حدود ۸۰ درصدی رسیدیم.

## کتابخانه‌های استفاده شده برای پیش‌پردازش

در پیش‌پردازش متن توییت‌ها از چند کتابخانه مهم حوزه NLP استفاده کردیم. در ادامه توضیح مختصری درباره هر یک از این کتابخانه‌ها را بررسی می‌کنیم.

**کتابخانه re** یکی از کتابخانه‌های استاندارد پایتون برای کار با عبارات منظم (regular expressions) است. این کتابخانه امکان جستجو، تطابق و جایگزینی الگوهای رشته‌ها را فراهم می‌کند.

**کتابخانه Hazm** نیز یکی از کتابخانه‌های محبوب برای پردازش زبان فارسی است. این کتابخانه ابزارهای مختلفی برای نرمال‌سازی، Tokenization، ریشه‌یابی و حذف واژگان غیرمفید فراهم می‌کند که در بخش‌های مربوط به پیش‌پردازش از آن استفاده شد.

در نهایت پیش‌پردازش روی متن توییت‌ها انجام شد و در ستون جدید cleaned\_tweet ذخیره شد:

<pre>data.loc[:, 'cleaned_tweet'] = data['tweet'].apply(preprocess_text)</pre>	<span>+ Code</span> <span>+ Markdown</span>																								
data.head()																									
<table border="1"> <thead> <tr> <th></th><th>tweet</th><th>emotion</th><th>cleaned_tweet</th></tr> </thead> <tbody> <tr> <td>0</td><td>دیشب خواب دیدم بمدی زدن نورش خیره کنده سیزه چیزی زدن نورش خیلی خیره کن</td><td>anger</td><td>... دیشب خواب دیدم بمدی زدن نورش خیره کنده سیزه ...</td></tr> <tr> <td>1</td><td>تری زدی ریشه‌ام، جوانه روید زخم راندی مرا دل</td><td>anger</td><td>... تری زدی ریشه‌ام، جوانه روید زخم راندی مرا دل ...</td></tr> <tr> <td>2</td><td>بدر سوخته‌ای بایام بهم میگه معنی داره خودم شرا</td><td>anger</td><td>... بدرو سوخته‌ای بایام بهم میگه معنی داره خودم شرا ...</td></tr> <tr> <td>3</td><td>شقیوید اخم نکنید . اقتدار نگاهات چشمانت . بیگار</td><td>anger</td><td>... شقیوید اخم نکنید . اقتدار نگاهات چشمانت . بیگار ...</td></tr> <tr> <td>4</td><td>مدح شادی غم نوشتنند برایت نوشتنند خنده لیت ، تصن</td><td>anger</td><td>... مدح شادی غم نوشتنند برایت نوشتنند خنده لیت ، تصن ...</td></tr> </tbody> </table>			tweet	emotion	cleaned_tweet	0	دیشب خواب دیدم بمدی زدن نورش خیره کنده سیزه چیزی زدن نورش خیلی خیره کن	anger	... دیشب خواب دیدم بمدی زدن نورش خیره کنده سیزه ...	1	تری زدی ریشه‌ام، جوانه روید زخم راندی مرا دل	anger	... تری زدی ریشه‌ام، جوانه روید زخم راندی مرا دل ...	2	بدر سوخته‌ای بایام بهم میگه معنی داره خودم شرا	anger	... بدرو سوخته‌ای بایام بهم میگه معنی داره خودم شرا ...	3	شقیوید اخم نکنید . اقتدار نگاهات چشمانت . بیگار	anger	... شقیوید اخم نکنید . اقتدار نگاهات چشمانت . بیگار ...	4	مدح شادی غم نوشتنند برایت نوشتنند خنده لیت ، تصن	anger	... مدح شادی غم نوشتنند برایت نوشتنند خنده لیت ، تصن ...
	tweet	emotion	cleaned_tweet																						
0	دیشب خواب دیدم بمدی زدن نورش خیره کنده سیزه چیزی زدن نورش خیلی خیره کن	anger	... دیشب خواب دیدم بمدی زدن نورش خیره کنده سیزه ...																						
1	تری زدی ریشه‌ام، جوانه روید زخم راندی مرا دل	anger	... تری زدی ریشه‌ام، جوانه روید زخم راندی مرا دل ...																						
2	بدر سوخته‌ای بایام بهم میگه معنی داره خودم شرا	anger	... بدرو سوخته‌ای بایام بهم میگه معنی داره خودم شرا ...																						
3	شقیوید اخم نکنید . اقتدار نگاهات چشمانت . بیگار	anger	... شقیوید اخم نکنید . اقتدار نگاهات چشمانت . بیگار ...																						
4	مدح شادی غم نوشتنند برایت نوشتنند خنده لیت ، تصن	anger	... مدح شادی غم نوشتنند برایت نوشتنند خنده لیت ، تصن ...																						

شکل ۷ اعمال پیش‌پردازش بر روی دیتاست

### ۳-۱. نمایش ویژگی

#### عملت استفاده از word embedding و روش های تولید آن

در پردازش زبان طبیعی، داده هایی که پردازش می شوند، متن خام هستند. با این حال، مدل ها فقط می توانند اعداد را پردازش کنند. با استفاده از word embedding کلمات با استفاده از یک vector نمایش داده می شوند به طوریکه کلماتی که مشابه هستند، در فضا نزدیک به یکدیگر قرار می گیرند. به این ترتیب مدل ها میتوانند زبان طبیعی را درک کنند و آموزش ببینند و عملکرد بهتری داشته باشند.

بطور کلی امبدینگ کلمات منجر می شود که وکتور کلماتی که معنای نزدیکی دارند یا با هم رابطه paradigmatic یا syntagmatic دارند در نزدیک یکدیگر در فضا قرار گیرند. این امر منجر می شود مدل ارتباط معنایی بین کلمات اسناد مختلف را درک کند و مثلا بتواند تسک classification انجام دهد و اسنادی که معنای یکسانی دارند را در یک کلاس قرار دهد.

به طور کلی دو دسته کلی برای نمایش بردارها وجود دارد:

**روش های Sparse:** در این روش های بردارها طول زیادی دارند، تعداد زیادی از درایه ها صفر هستند و سنتی تر هستند. این روش ها معمولا بر اساس شمارش کلمات میباشد. در این روش های بردارها طول زیادی دارند، تعداد زیادی از درایه ها صفر هستند و سنتی تر هستند. این روش ها معمولا بر اساس شمارش کلمات میباشد. انواع این روش ها شامل tf که بر اساس تعداد کلماتی که در اطراف یک کلمه هستند بردار آن تعریف می شود، tf-idf که میزان جنرال یا خاص بودن یک کلمه را نیز در idf در نظر می گیرد، PPMI که ماتریس word-word داده ها را تشکیل میدهد.

**روش های Dense:** در این روش ها بردارها طول کوتاهی دارند، معمولا صفر ندارند و معمولا برای ایجاد امبدینگ در این روش ها از یک classifier استفاده می شود. **Fasttext**، **GloVe**، **word<sup>2</sup>vec** و **BERT** از جمله نمونه های این روش هستند.

اما روش هایی که در بالا بیان شد، Contextual نیستند یعنی برای هر کلمه یک وکتور در تمامی context ها در نظر می گیرند. مثلا شیر خوارکی با شیر جنگل یکسان است. اما روش هایی دیگر نظیر transformer یا BERT ها نیز وجود دارند که در این روش ها با توجه به context هر کلمه در context های مختلف، امبدینگ آن مشخص می شود. در این روش های از مفهوم self attention استفاده می شود و امبدینگ هر کلمه در هر جمله مشخص می شود.

## استفاده از توکن‌ساز ParsBERT

در این بخش ابتدا مدل توکن‌ساز از پیش‌آموزش دیده شده‌ی ParsBERT را فراخوانی کرده و برای تبدیل داده‌های متنی پیش‌پردازش شده به اعداد استفاده می‌کنیم. همچنین برای یکسان‌سازی طول تمام سطرها از Padding استفاده کرده و طول تمام سطرها را برابر با ۳۲ تنظیم می‌کنیم. یک نمونه از اعمال این توکن‌ساز را در شکل زیر مشاهده می‌کنیم:

شکا، ۸ اعمال، توکن‌ساز؛ PARSBERT

Input IDs نشان‌دهنده شناسه‌های عددی مربوط به توکن‌های توييت‌ها هستند. اين شناسه‌ها از Vocabulary مدل ParsBERT استخراج مي‌شوند.

Attention Masks نشان‌دهنده ماسک‌های توجه هستند که به مدل اطلاع می‌دهند که کدام توکن‌ها باید مورد توجه قرار گیرند و کدام‌ها باید نادیده گرفته شوند (ید‌ها).

## بردار تعبیه یا Embedding

در این بخش، از مدل از پیش آموزش دیده ParsBERT برای دریافت بردارهای تعبیه (embeddings) استفاده می‌کنیم. به این منظور ابتدا مدل ParsBERT را بارگذاری کرده و سپس توابع مورد نیاز برای مدیریت حافظه و پردازش داده‌ها به صورت batch را تعریف می‌کنیم. هر batch شامل ۶۴ نمونه بوده و پس از پردازش هر batch، بردارهای تعبیه به حافظه اصلی منتقل شده و حافظه GPU پاکسازی می‌شود.

## تکنیک‌های مدیریت حافظه

تکنیک‌های مدیریت حافظه مانند تکه‌تکه کردن (batch processing) و استفاده از کتابخانه `gc` برای جمع‌آوری زباله (garbage collection) به ما کمک می‌کنند تا با محدودیت‌های حافظه به خوبی کنار بیاییم و از حافظه به صورت بهینه استفاده کنیم. در اجرای بدون استفاده از این تکنیک‌ها، محیط Kaggle خطای پرشدن حافظه می‌دهد.

تکه‌تکه کردن یا Batch Processing یک تکنیک موثر برای مدیریت حافظه هنگام پردازش داده‌های بزرگ است. در این روش، داده‌ها به دسته‌های کوچکتر تقسیم می‌شوند و هر دسته به صورت جداگانه پردازش می‌شود در نتیجه این کار باعث می‌شود که نیاز به حافظه در هر لحظه کاهش یابد. در کدی که ارائه شده، داده‌ها به دسته‌هایی با حجم ۶۴ نمونه تقسیم شدند و هر دسته به صورت جداگانه پردازش شد.

## کتابخانه `gc` برای Garbage Collection

کتابخانه `gc` در پایتون برای مدیریت حافظه و جمع‌آوری زباله استفاده می‌شود. این کتابخانه به طور خودکار Object‌هایی را که دیگر به آن‌ها نیازی نیست از حافظه حذف می‌کند تا حافظه آزاد شود. در کد مربوط به این قسمت، پس از پردازش هر دسته، از `gc.collect()` برای جمع‌آوری زباله و `torch.cuda.empty_cache()` برای پاکسازی حافظه GPU استفاده شد.

پس از پردازش تمام دسته‌ها، بردارهای تعبیه مربوط به هر دسته ترکیب می‌شود تا ماتریس نهایی با ابعاد (۱۲۰, ۳۲, ۱۱۳۸۲۹) تشکیل شود.

```
batch_size = 64
embeddings = get_embeddings_in_batches(data, tokenizer, model, MAX_LEN, batch_size, device)

print("All embeddings shape:", embeddings.shape)

All embeddings shape: torch.Size([113829, 32, 120])
```

شکل ۹ استخراج تعبیه‌ها به صورت دسته‌ای

این خروجی نشان می‌دهد که ۱۱۳۸۲۹ نمونه، هر کدام با طول ۳۲ و ابعاد بردار امبدینگ ۱۲۰ است. در نهایت از این ماتریس برای ورودی مدل استفاده خواهیم کرد.

حال به بررسی سوالات مطرح شده در انتهای این بخش می‌پردازیم. هرچند که در توضیحات اولیه درباره امبدینگ در ابتدای همین بخش توضیحات کاملی در این باره ارائه شد.

### ابعاد پیشفرض بردار تعبیه در ParsBERT چقدر است؟

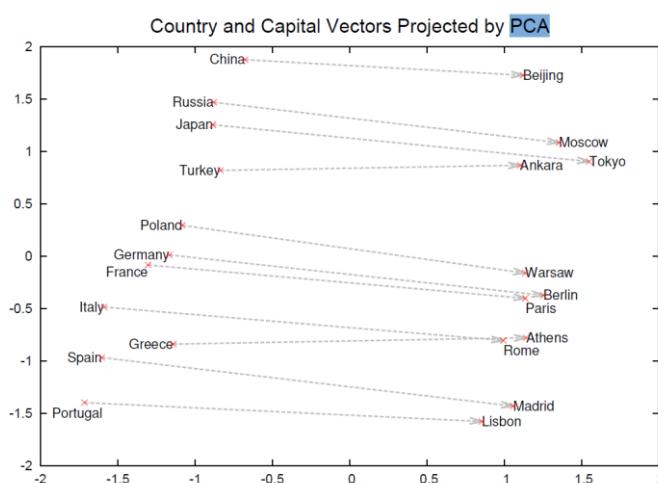
ابعاد پیشفرض بردار تعبیه (embedding) در مدل ParsBERT برابر با ۷۶۸ است.

### تعداد ابعاد این بردار بیانگر چیست؟

عداد ابعاد بردار تعبیه (در اینجا ۷۶۸) بیانگر تعداد ویژگی‌هایی است که هر کلمه در فضای برداری دارد. یعنی هر کلمه با ۷۶۸ عدد نمایش داده می‌شود. این ویژگی‌ها شامل جنبه‌های مختلف معنایی، نحوی و گرامری کلمات هستند. هر بعد از این فضا می‌تواند نمایانگر یک ویژگی خاص باشد. به طور کلی، هر بعد از این فضای برداری یک ویژگی از کلمه را در بر دارد و ترکیب این ویژگی‌ها باعث می‌شود که مدل بتواند معنای کلمات را به صورت برداری درک کند.

مفهوم بردار تعبیه را توضیح دهید و بیان کنید به نظر شما کدام یک از کلمات موجود در مجموعه داده ممکن است تعبیه نزدیک به هم داشته باشند؟ (توضیحات کامل ابتدای بخش ۳)

بردار تعبیه یا Word Embedding روشی برای نمایش کلمات به صورت بردارهای عددی است. در داده‌های متنی، کلماتی که معانی مشابه یا کاربردهای مشابهی دارند، معمولاً بردارهای تعبیه نزدیک به هم دارند. برای مثال کلمات **شاد** و **خوشحال** احتمالاً بردارهای تعبیه نزدیک به هم خواهند داشت زیرا هر دو بیانگر یک حالت احساسی مثبت هستند. کلمات **غمگین** و **ناراحت** نیز احتمالاً بردارهای تعبیه مشابهی خواهند داشت چون هر دو بیانگر یک حالت احساسی منفی هستند.



شکل ۱۰ فضای بردار کلمات - تبدیل شده به ۲ بعد

برای مثال مشاهده میشود که در تصویر بالا، بدون آنکه به مدل یاد داده شود که پایتحت چیست کاملا مشخص شده که پایتحت ها یک سمت و کشورها سمت دیگر هستند و بردار فاصله این دو به یکدیگر موازی هستند.

## ۴-۱. ساخت مدل

### تقسیم داده‌ها

در این بخش، داده‌ها را به مجموعه‌های آموزشی، اعتبارسنجی و تست تقسیم می‌کنیم. نسبت تقسیم ۷۰ به ۳۰ است. و در نهایت ۲۰٪ از داده‌های آموزشی را برای اعتبارسنجی کنار می‌گذاریم. قبل از اینکه داده‌ها تقسیم شوند، لیبل‌های متنی ستون احساسات را به فرمت عددی تبدیل می‌کنیم.

```
Split Data (70 - 30)

+ Code + Markdown

labels = data['emotion'].factorize()[0]

train_embeddings, test_embeddings, train_labels, test_labels = train_test_split(embeddings, labels, test_size=0.3, random_state=42)
train_embeddings, val_embeddings, train_labels, val_labels = train_test_split(train_embeddings, train_labels, test_size=0.2, random_state=42)
```

شکل ۱۱ تقسیم داده‌ها به مجموعه‌های آموزشی، اعتبارسنجی و تست

پس از تقسیم داده‌ها، داده‌های تعبیه شده و برچسب‌ها را به تنسورهای PyTorch تبدیل کرده و سپس دیتاست‌های آموزشی، اعتبارسنجی و تست را ایجاد می‌کنیم و از این نظر اطمینان حاصل می‌کنیم که داده‌های تست در مراحل آموزش مشاهده نخواهند شد.

```
Create Datasets

train_embeddings_tensor = train_embeddings.clone().detach()
train_labels_tensor = torch.tensor(train_labels).clone().detach()

val_embeddings_tensor = val_embeddings.clone().detach()
val_labels_tensor = torch.tensor(val_labels).clone().detach()

test_embeddings_tensor = test_embeddings.clone().detach()
test_labels_tensor = torch.tensor(test_labels).clone().detach()

train_dataset = TensorDataset(train_embeddings_tensor, train_labels_tensor)
val_dataset = TensorDataset(val_embeddings_tensor, val_labels_tensor)
test_dataset = TensorDataset(test_embeddings_tensor, test_labels_tensor)
```

شکل ۱۲ ساخت دیتاست‌های آزمون، اعتبارسنجی و تست

در کد بالا، از `clone().detach()` استفاده برای تبدیل بردارهای تعبیه و لیبل‌ها به تنسورهای PyTorch شده است. این کار به ما کمک می‌کند تا کپی‌های مستقلی از داده‌ها ایجاد کنیم که با گراف محاسباتی مرتبط نباشند.

باتوجه به اینکه در این پرسش در حلقه‌ی جستجوی حریصانه Batch با اندازه مختلف آزمایش می‌شود، یک تابع برای ساخت Dataloader بر حسب اندازه Batch پیاده‌سازی می‌کنیم. خروجی این تابع هم آزمایش شد و به خوبی دیتالودر ساخته شد. نمونه خروجی:

Sample embedding shape: `torch.Size([64, 32, 120])`

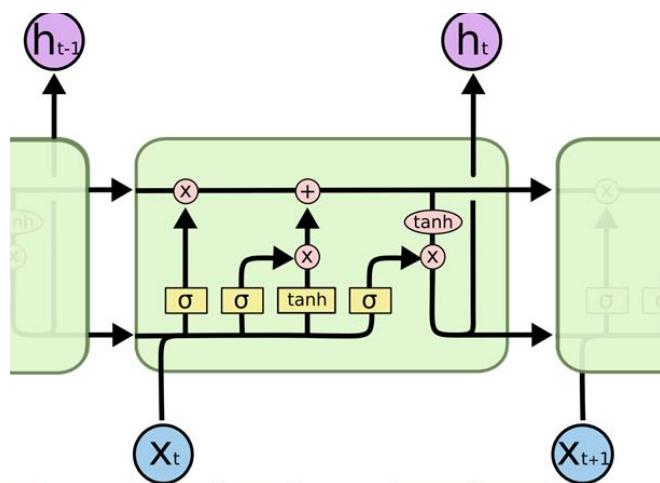
Sample label shape: `torch.Size([64])`

عدد اول بردار اندازه Batch عدد دوم اندازه Sequence که ۳۲ تایی است و عدد سوم هم ابعاد بردار تعبیه است.

قبل از اینکه به بررسی مدل ارائه شده در مقاله بپردازیم نگاهی به معماری LSTM خواهیم داشت.

### معماری LSTM

LSTM نوعی از RNN است که چهار لایه دارد. این نوع شبکه وابستگی‌های بلند مدت را در `sequence` های داده‌ها یاد می‌گیرد. LSTM دارای ۳ گیت است که در تصویر زیر مشخص شده است:



شکل ۱۳ معماری LSTM

این گیت‌ها به ترتیب گیت `forget`، گیت `input` و گیت `output` می‌باشند. همچنین همانطور که در تصویر نشان داده شده است، در LSTM یک خط افقی از بالای نمودار عبور می‌کند که در واقع `cell state` می‌باشد. این خط همچون یک نقاله در LSTM وجود دارد. این معماری LSTM را قادر می‌سازد که با استفاده از گیت `input`، مقداری را از بر روی `cell state` قرار میدهد. با استفاده از گیت `forget`، مقداری را از روی `cell state` حذف می‌کند و با استفاده از گیت `output` مقداری از روی `cell state` برمی‌دارد و استفاده می‌کند. این عملیات‌ها را LSTM با استفاده از تابع `sigmoid` انجام میدهد.

مزایا:

- همانطوریکه توضیح دادیم، با توجه به ساختاری که معماری LSTM دارد، میتواند وابستگی های طولانی ( long-term dependencies ) را در دیتاهای Recurrent بیشتر از مدل های سنتی RNN هندل کند.
  - در برابر مشکل vanishing gradient نسبت به مدل های سنتی RNN بہتر عمل می کند.
  - LSTM میتواند ساختارها و روابط پیچیده تری را نسبت به مدل ها سنتی RNN مدل کند و یاد بگیرد. در هنگام استفاده از مجموعه دادهها با توالی طولانی تر، LSTM دقیق تر است.
- مدل CNN-LSTM: توضیح و پیاده‌سازی با توجه به مقاله**
- مدل CNN-LSTM ترکیبی از شبکه‌های عصبی CNN و شبکه‌های LSTM است. این مدل در مقاله برای تحلیل احساسات توییت‌های سیاسی فارسی به کار رفته است.

Layer (type:depth-idx)	Output Shape	Param #
=====		
CNNLSTM	[64, 6]	--
Conv1d: 1-1	[64, 32, 32]	11,552
MaxPool1d: 1-2	[64, 32, 16]	--
LSTM: 1-3	[64, 16, 100]	53,600
Dropout: 1-4	[64, 100]	--
Linear: 1-5	[64, 6]	606
=====		
Total params:	65,758	
Trainable params:	65,758	
Non-trainable params:	0	
Total mult-adds (M):	78.58	
=====		
Input size (MB):	0.98	
Forward/backward pass size (MB):	1.35	
Params size (MB):	0.26	
Estimated Total Size (MB):	2.59	
=====		

شکل ۱۴ معماری مدل CNN-LSTM

لایه Conv1D: این لایه به دنبال استخراج ویژگی‌ها از داده‌های ورودی است. اندازه فیلتر برابر با ۳ و تعداد فیلترها برابر با ۳۲ است.

لایه MaxPool1D: این لایه اندازه داده‌ها را کاهش می‌دهد و ویژگی‌های مهم‌تر را انتخاب می‌کند.

لایه LSTM: این لایه داده‌های CNN را پردازش کرده و الگوهای ترتیبی را یاد می‌گیرد.

لایه Dropout: برای جلوگیری از overfitting استفاده می‌شود و نرخ آن برابر با ۰.۱ است.

لایه Linear: خروجی نهایی مدل را تولید می‌کند.

ترکیب این دو مدل شبکه باعث می‌شود که مدل بتواند هم ویژگی‌های محلی و هم الگوهای ترتیبی در جملات را به خوبی استخراج کند.

پس از پیاده‌سازی مدل، حلقه Train نیز پیاده‌سازی شد. این حلقه در این پرسشن به نحوی پیاده‌سازی شده که هر بار بتوانیم با چندین پارامتر مدل را آموزش داده و بهترین مدل را بتوانیم با توجه به خروجی آن انتخاب کنیم. با توجه به جدول ارائه شده در مقاله ازتابع CrossEntropyLoss، LOSS استفاده می‌کنیم. در انتهای حلقه نیز نمودار مربوط به خطای دقت بر حسب Epoch را نمایش می‌دهیم.

### جستجوی حریصانه پارامترهای بهینه

در این بخش، از روش جستجوی حریصانه برای تنظیم پارامترهای بهینه مدل استفاده می‌کنیم. بتدا فضای جستجو را تعریف کرده و سپس با استفاده از حلقه‌های تو در تو تمامی ترکیبات ممکن از پارامترها را بررسی می‌کنیم. در هر تکرار، مدل آموزش داده شده و عملکرد آن بر روی مجموعه اعتبارسنجی ارزیابی می‌شود. در نهایت، بهترین ترکیب پارامترها که کمترین خطای داشته باشد، انتخاب می‌شود. فضای جستجو به صورت زیر است:

```
batch_sizes = [8, 64]
learning_rates = [0.001, 0.0001]
optimizers = [Adam, SGD]
```

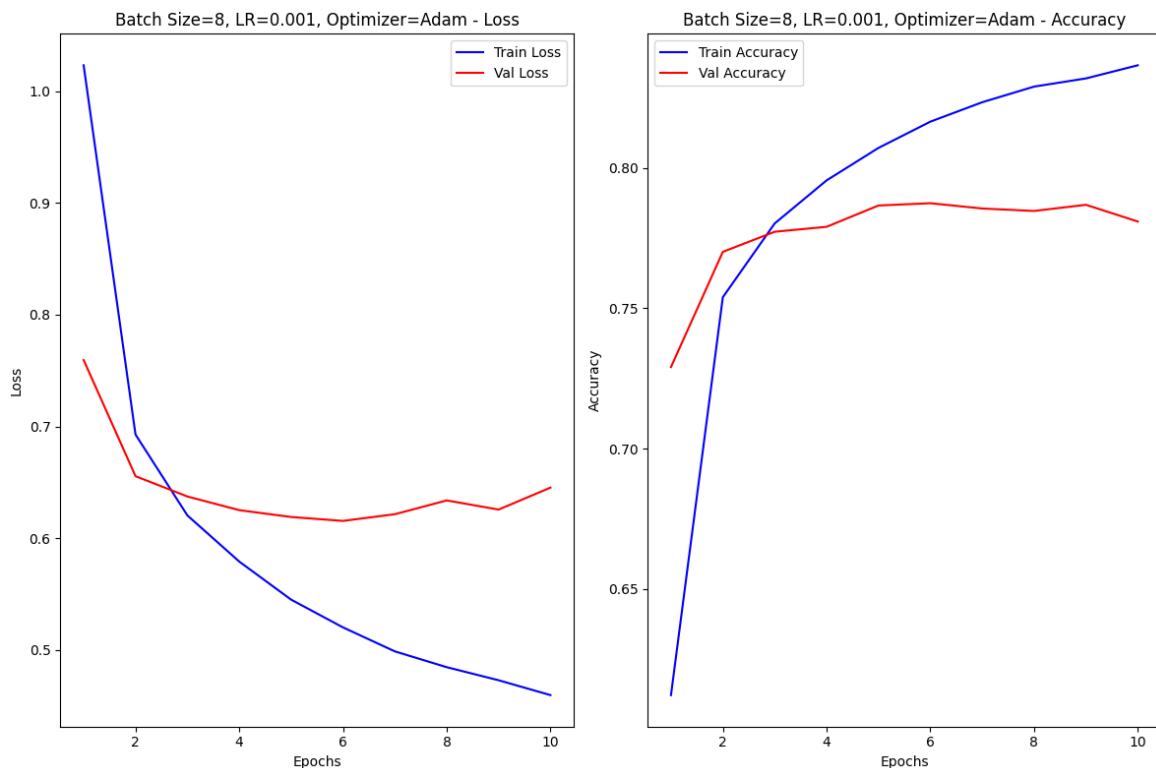
شکل ۱۵ فضای جستجوی حریصانه

در نهایت مدل بر روی ۸ حالت مختلف هر کدام به اندازه ۱۰ ایپاک، آموزش دیده شد. حالت اول:

```
Training with batch_size=8, learning_rate=0.001, optimizer=Adam
Epoch 1/10, Train Loss: 1.0231, Train Accuracy: 0.6122, Val Loss: 0.7594, Val Accuracy: 0.7290
Epoch 2/10, Train Loss: 0.6927, Train Accuracy: 0.7539, Val Loss: 0.6555, Val Accuracy: 0.7701
Epoch 3/10, Train Loss: 0.6204, Train Accuracy: 0.7802, Val Loss: 0.6373, Val Accuracy: 0.7772
Epoch 4/10, Train Loss: 0.5791, Train Accuracy: 0.7955, Val Loss: 0.6251, Val Accuracy: 0.7790
Epoch 5/10, Train Loss: 0.5450, Train Accuracy: 0.8071, Val Loss: 0.6189, Val Accuracy: 0.7866
Epoch 6/10, Train Loss: 0.5202, Train Accuracy: 0.8164, Val Loss: 0.6154, Val Accuracy: 0.7874
Epoch 7/10, Train Loss: 0.4987, Train Accuracy: 0.8234, Val Loss: 0.6214, Val Accuracy: 0.7855
Epoch 8/10, Train Loss: 0.4846, Train Accuracy: 0.8289, Val Loss: 0.6338, Val Accuracy: 0.7846
Epoch 9/10, Train Loss: 0.4729, Train Accuracy: 0.8318, Val Loss: 0.6256, Val Accuracy: 0.7868
Epoch 10/10, Train Loss: 0.4596, Train Accuracy: 0.8365, Val Loss: 0.6452, Val Accuracy: 0.7809
```

شکل ۱۶ نتیجه آموزش مدل برای حالت اول

تمام حالات در خروجی نوت بوک پیوست شده قابل مشاهده هستند. در ادامه به بهترین حالت می‌پردازیم که کمترین خطای را داشته است. این مورد در اجراهای مختلف متفاوت بود. خروجی نوت بوک فعلی برابر است با:



شکل ۱۷ نمودار خطای دقت نسبت به epoch برای حالت اول- دقت اعتبارسنجی حدود ۷۸ درصد

Best Hyperparameters: Batch Size=8, Learning Rate=0.001, Optimizer=Adam

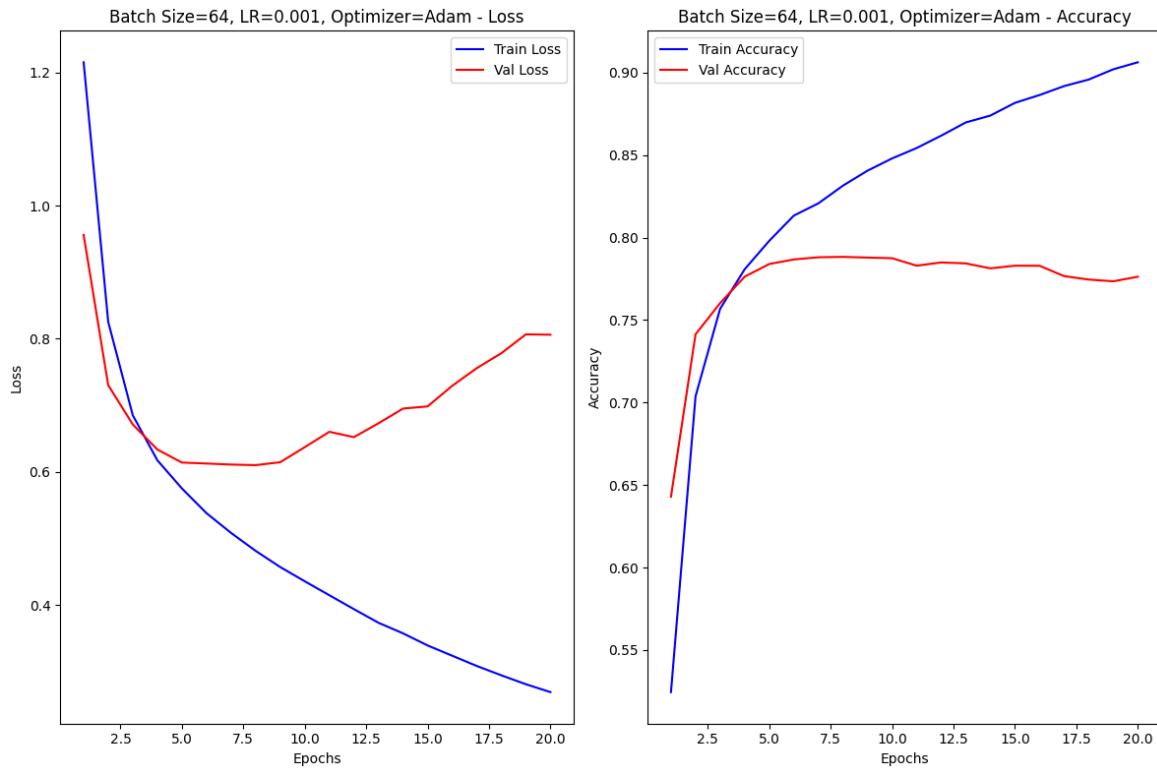
ولی در بیشتر اوقات مقدار Batch برابر ۶۴ خروجی بهتری داشت. پس مدل را مجدداً با این مقدار آموزش دادیم و با ۲۰ ایپاک مقدار خطای کمتری نسبت به ۸ حاصل شد. با توجه به این مورد، ادامه پرسش را با Batch برابر ۶۴ ادامه دادیم. نتیجه آموزش به صورت زیر است :

```

Epoch 1/20, Train Loss: 1.2153, Train Accuracy: 0.5245, Val Loss: 0.9559, Val Accuracy: 0.6429
Epoch 2/20, Train Loss: 0.8253, Train Accuracy: 0.7038, Val Loss: 0.7302, Val Accuracy: 0.7413
Epoch 3/20, Train Loss: 0.6850, Train Accuracy: 0.7568, Val Loss: 0.6712, Val Accuracy: 0.7602
Epoch 4/20, Train Loss: 0.6174, Train Accuracy: 0.7888, Val Loss: 0.6336, Val Accuracy: 0.7763
Epoch 5/20, Train Loss: 0.5751, Train Accuracy: 0.7980, Val Loss: 0.6140, Val Accuracy: 0.7839
Epoch 6/20, Train Loss: 0.5379, Train Accuracy: 0.8133, Val Loss: 0.6125, Val Accuracy: 0.7880
Epoch 7/20, Train Loss: 0.5082, Train Accuracy: 0.8207, Val Loss: 0.6110, Val Accuracy: 0.7880
Epoch 8/20, Train Loss: 0.4813, Train Accuracy: 0.8314, Val Loss: 0.6101, Val Accuracy: 0.7883
Epoch 9/20, Train Loss: 0.4570, Train Accuracy: 0.8405, Val Loss: 0.6145, Val Accuracy: 0.7878
Epoch 10/20, Train Loss: 0.4356, Train Accuracy: 0.8479, Val Loss: 0.6371, Val Accuracy: 0.7875
Epoch 11/20, Train Loss: 0.4147, Train Accuracy: 0.8542, Val Loss: 0.6602, Val Accuracy: 0.7829
Epoch 12/20, Train Loss: 0.3938, Train Accuracy: 0.8617, Val Loss: 0.6522, Val Accuracy: 0.7849
Epoch 13/20, Train Loss: 0.3733, Train Accuracy: 0.8697, Val Loss: 0.6730, Val Accuracy: 0.7843
Epoch 14/20, Train Loss: 0.3574, Train Accuracy: 0.8739, Val Loss: 0.6951, Val Accuracy: 0.7813
Epoch 15/20, Train Loss: 0.3392, Train Accuracy: 0.8816, Val Loss: 0.6983, Val Accuracy: 0.7829
Epoch 16/20, Train Loss: 0.3238, Train Accuracy: 0.8863, Val Loss: 0.7292, Val Accuracy: 0.7829
Epoch 17/20, Train Loss: 0.3084, Train Accuracy: 0.8918, Val Loss: 0.7558, Val Accuracy: 0.7766
Epoch 18/20, Train Loss: 0.2944, Train Accuracy: 0.8957, Val Loss: 0.7783, Val Accuracy: 0.7745
Epoch 19/20, Train Loss: 0.2809, Train Accuracy: 0.9019, Val Loss: 0.8066, Val Accuracy: 0.7735
Epoch 20/20, Train Loss: 0.2691, Train Accuracy: 0.9062, Val Loss: 0.8060, Val Accuracy: 0.7762

```

شکل ۱۸ نتیجه آموزش بر اساس بهترین پارامترها



شکل ۹ نمودار خطا و دقت نسبت به epoch برای بهترین حالت- دقت اعتبارسنجی حدود ۷۹ درصد

خطای آموزشی به تدریج کاهش یافته و به مقادیر پایینی رسیده است که نشان‌دهنده بمبود مدل در طول آموزش است. خطای اعتبارسنجی ابتدا کاهش یافته و سپس افزایش یافته است که نشان‌دهنده امکان overfitting در مدل است. دقت اعتبارسنجی نیز ابتدا افزایش یافته و سپس ثبیت شده است که نشان‌دهنده عملکرد خوب مدل بر روی داده‌های اعتبارسنجی است. پس به طور کلی در حدود ۱۰ اجرای متفاوت مقادیر زیر کمترین خطای اعتبارسنجی را داشت:

### Best Hyperparameters: Batch Size=۶۴, Learning Rate=۰،۰۰۱، Optimizer=Adam

در ادامه با این پارامترها، مدل‌های CNN و LSTM ساده را آموزش می‌دهیم:

#### مدل CNN ساده

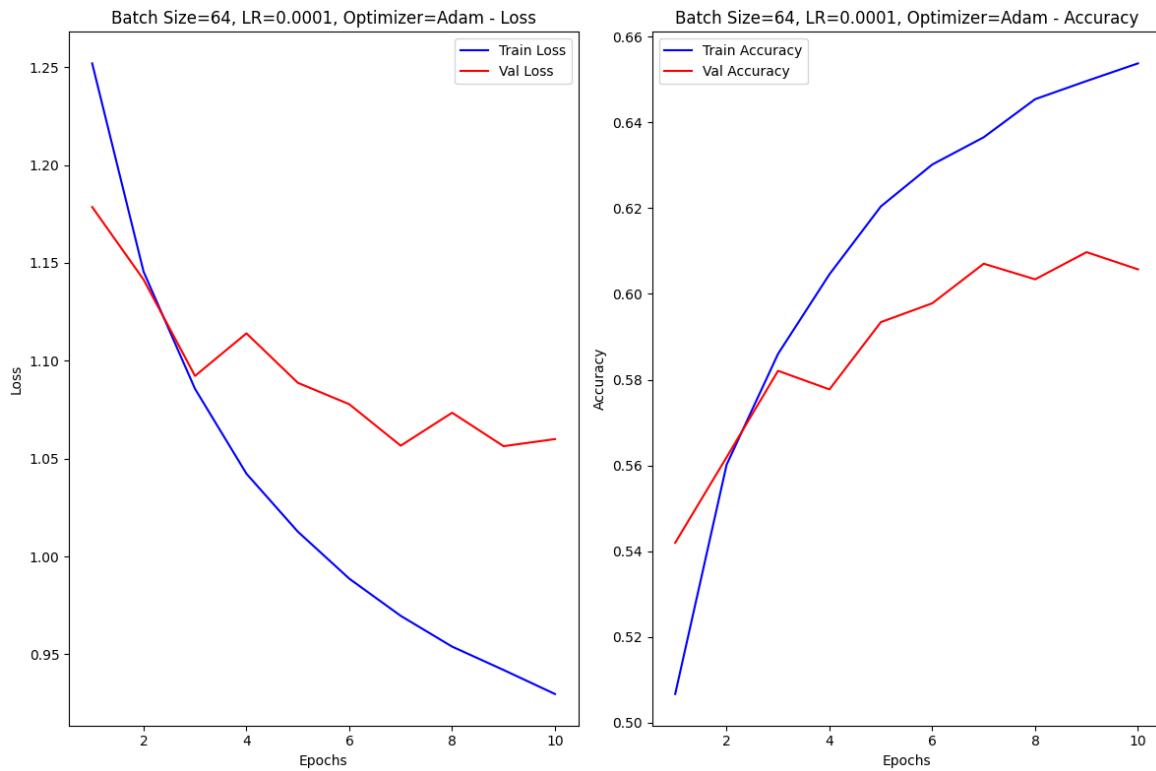
معماری این مدل در شکل زیر مشاهده می‌شود. این مدل از لایه‌های CNN و یک لایه خطی تشکیل شده است که توضیحات مربوط به این لایه‌ها را در قسمت قبلی با معرفی معماری CNN-LSTM به طور کامل بررسی کردیم.

```

=====
Layer (type:depth-idx)           Output Shape        Param #
=====
SimpleCNN
├─Conv1d: 1-1                   [64, 6]             --
├─MaxPool1d: 1-2                [64, 32, 32]       11,552
├─Linear: 1-3                  [64, 32, 16]       --
├─Linear: 1-4                  [64, 100]          51,300
├─Dropout: 1-4                 [64, 100]          --
└─Linear: 1-5                  [64, 6]            606
=====
Total params: 63,458
Trainable params: 63,458
Non-trainable params: 0
Total mult-adds (M): 26.98
=====
Input size (MB): 0.98
Forward/backward pass size (MB): 0.58
Params size (MB): 0.25
Estimated Total Size (MB): 1.82
=====
```

شکل ۲۰ معماری مدل CNN ساده

نتیجه آموزش با پارامترهای بهینه به صورت زیر است:



شکل ۲۱ نمودار خطا و دقت نسبت به epoch برای مدل CNN ساده

همانطور که مشاهده می شود، دقت این مدل ( حدود ۶۰ درصد برای داده اعتبارسنجی) بسیار پایین تر از مدل پیشنهادی است و دلیل آن ماهیت CNN است که نمی تواند ماهیت ترتیبی بودن کلمات جمله که امر مهمی در تحلیل احساسات است را به خوبی یاد بگیرد.

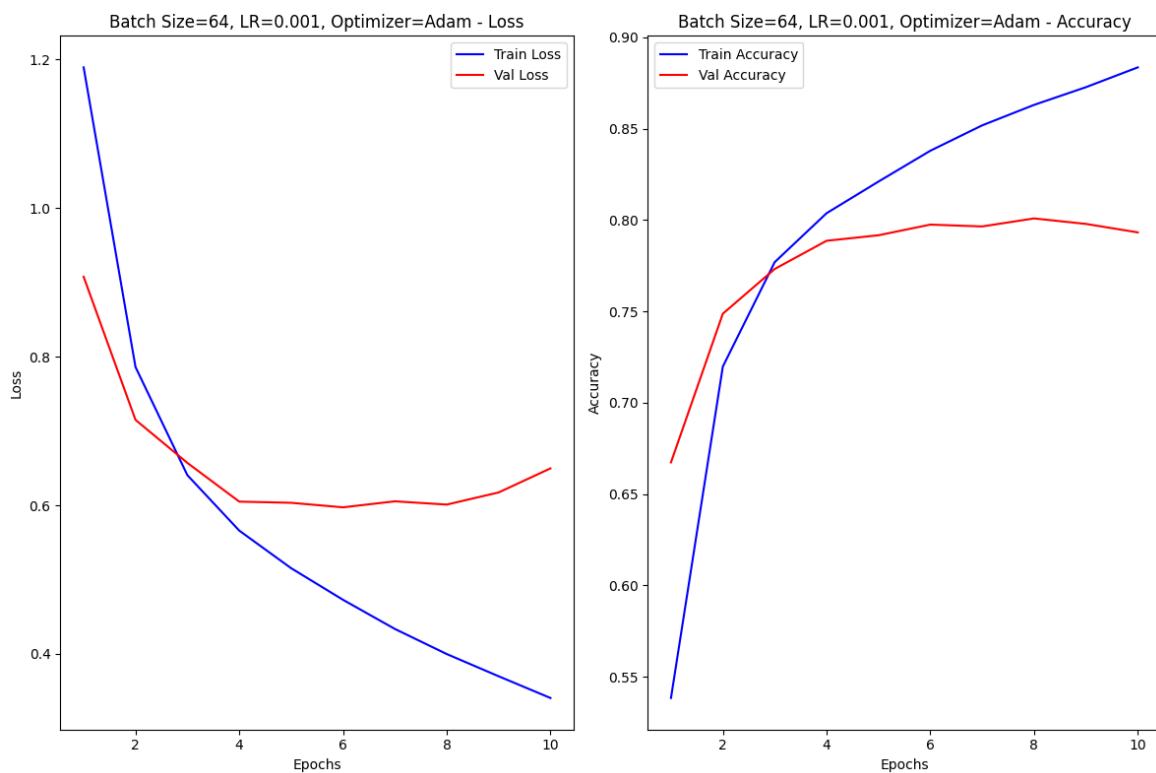
## مدل LSTM ساده

معماری این مدل در شکل زیر مشاهده می‌شود. توضیحات مربوط به این لایه‌ها را در قسمت قبلی با معرفی معماری LSTM به طور کامل بررسی کردیم.

```
=====
Layer (type:depth-idx)          Output Shape       Param #
=====
SimpleLSTM
|---LSTM: 1-1                  [64, 6]           --
|---Dropout: 1-2                [64, 32, 100]     88,800
|---Linear: 1-3                 [64, 100]          --
|---Linear: 1-3                 [64, 6]            606
=====
Total params: 89,406
Trainable params: 89,406
Non-trainable params: 0
Total mult-adds (M): 181.90
=====
Input size (MB): 0.98
Forward/backward pass size (MB): 1.64
Params size (MB): 0.36
Estimated Total Size (MB): 2.98
=====
```

شکل ۲۲ معماری مدل LSTM ساده

نتیجه آموزش با پارامترهای بهینه به صورت زیر است:



شکل ۲۳ نمودار خطا و دقت نسبت به epoch برای مدل LSTM ساده

همانطور که مشاهده می‌شود، دقت این مدل (۸۰ درصد برای داده اعتبارسنجی) بسیار شبیه به مدل CNN-LSTM است و حدوداً ۱ درصد بیشتر از آن است. مدل به خوبی توانست الگوهای موجود در داده‌ها را یاد بگیرد و عملکرد قابل قبولی را نشان دهد.

انتظار داشتیم با توجه به معما ر ارائه شده در مقاله، این مدل نتیجه ضعیف‌تری نسبت به مدل ترکیبی CNN-LSTM داشته باشد. ولی در عمل برای این دیتا است این نتیجه حاصل شد. البته باید توجه داشته باشیم با توجه به اینکه اعلام شده دیتا است این سوال متفاوت بوده و همچنین ما از فضای حالتی استفاده کرده‌ایم که روی دیتا است مقاله جواب بهینه داده، پس ممکن است مقادیر های پر پارامتری وجود داشته باشد که این مدل‌ها دقت بالاتری داشته باشند.

### بررسی نقاط ضعف و قوت مدل‌ها

CNN مدل	
استخراج ویژگی‌های محلی: CNN در استخراج ویژگی‌های محلی از داده‌های متنی بسیار قوی است.	نقاط قوت
مقاومت در برابر نویز: فیلترهای کانولوشنی و عملیات MaxPooling می‌توانند نویز را کاهش داده و ویژگی‌های مهم‌تر را انتخاب کنند.	نقاط ضعف
عدم درک روابط ترتیبی: CNN به خوبی نمی‌تواند روابط ترتیبی Long term (بلند مدت) را شناسایی کند. این موضوع در پردازش زبان طبیعی که روابط بین کلمات بسیار مهم است، محدودیت محسوب می‌شود.	نقاط ضعف
تعداد پارامترهای بالا: تعداد پارامترهای شبکه‌های CNN به خصوص در لایه‌های عمیق‌تر می‌تواند بسیار زیاد باشد که باعث افزایش زمان آموزش و نیاز به حافظه بیشتر می‌شود.	نقاط ضعف

LSTM مدل	
درک روابط ترتیبی بلندمدت: LSTM می‌تواند روابط ترتیبی بلندمدت و واپس‌گردی‌های طولانی‌مدت بین کلمات را به خوبی شناسایی کند.	نقاط قوت
حافظه طولانی‌مدت: LSTM قادر است اطلاعات را برای مدت طولانی در حافظه خود نگه دارد و از آن برای پیش‌بینی‌های دقیق‌تر استفاده کند.	نقاط ضعف

## نقاط ضعف

پایین بودن کارایی در استخراج ویژگی‌های محلی: LSTM در استخراج ویژگی‌های محلی به اندازه CNN قوی نیست و ممکن است نتواند الگوهای محلی را به خوبی شناسایی کند.

نیاز به زمان آموزش بیشتر: به دلیل پیچیدگی‌های موجود در LSTM، زمان آموزش این مدل‌ها معمولاً بیشتر است.

## ادغام هردو مدل

ادغام مدل‌های LSTM و CNN به منظور استفاده از نقاط قوت هر دو مدل و جبران نقاط ضعف آن‌ها انجام می‌شود. CNN می‌تواند ویژگی‌های محلی و الگوهای کوچک را به خوبی استخراج کند. این ویژگی‌ها می‌توانند به عنوان ورودی به شبکه LSTM داده شوند. LSTM می‌تواند روابط ترتیبی بلندمدت و وابستگی‌های زمانی بین ویژگی‌های استخراج شده توسط CNN را یاد بگیرد. این امر کمک می‌کند که مدل الگوهای پیچیده‌تری را شناسایی کند.

ولی در این دیتابست ما نتیجه مانند مقاله دریافت نشد. در ادامه دلایل احتمالی را بررسی می‌کنیم.

## تowییت‌های سیاسی در مقابل towییت‌های عمومی

تowییت‌های سیاسی معمولاً شامل واژگان و اصطلاحات خاصی هستند که ممکن است الگوهای محلی مانند نام افراد، احزاب، یا اصطلاحات سیاسی را بیشتر نشان دهند. این الگوها ممکن است با استفاده از CNN به خوبی استخراج شوند. همچنین این towییت‌ها ممکن است دارای ساختار محتوای پیچیده‌تری باشند که نیاز به استخراج ویژگی‌های محلی دارند. به عنوان مثال، ذکر نام یک سیاستمدار و سپس بیان یک نظر.

از طرفی towییت‌های عمومی معمولاً دارای ساختار ساده‌تری هستند و ممکن است نیاز به استخراج ویژگی‌های محلی پیچیده نداشته باشند. تنوع موضوعی بالا در towییت‌های عمومی می‌تواند باعث کاهش کارایی CNN شود، زیرا الگوهای محلی کمتری برای استخراج وجود دارد.

در ادامه به طور دقیق‌تر نتایج مدل‌های مختلف را باهم مقایسه می‌کنیم.

## ۱-۵. ارزیابی

در این بخش مقادیر دقت، precision و recall را برای هر مدل بررسی خواهیم کرد که فرمول های آنها به صورت زیر است:

:**Accuracy** معیار

$$\text{classification Accuracy} = \frac{\text{Total number of correct decisions}}{\text{Total number of decisions made}}$$

ایراد این روش این است که ارزش کلاس ها در نظر گرفته نمیشود مثلا اگر کلاس ها imbalance بوده و توازن نداشته باشند ممکن است accuracy بالایی داشته باشیم اما مدلمان ضعیف باشد.

:**Precision** معیار

$$P = \frac{TP}{TP + FP}$$

:**Recall** معیار

$$R = \frac{TP}{TP + FN}$$

:**F\_1** معیار

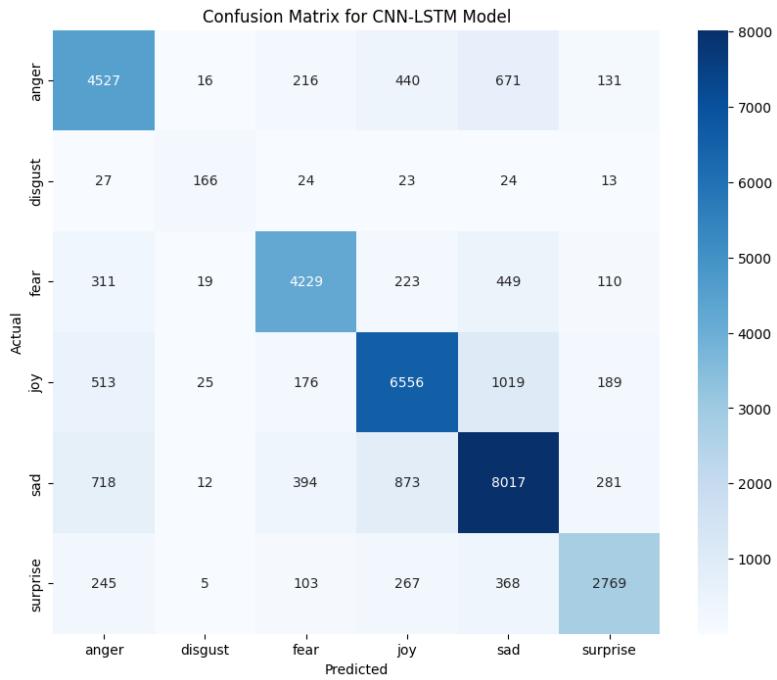
$$F_1 = \frac{2P \times R}{P + R}$$

در این فرمول p برابر با recall و R برابر با precision است.

خروجی داده های تست برای هر مدل:

:**CNN-LSTM** ارزیابی مدل

CNN-LSTM Model:  
Accuracy: 0.7691  
Precision: 0.7591  
Recall: 0.7391  
F1-score: 0.7483



شکل ۲۴ ماتریس آشتفتگی مدل CNN-LSTM

:CNN ارزیابی مدل

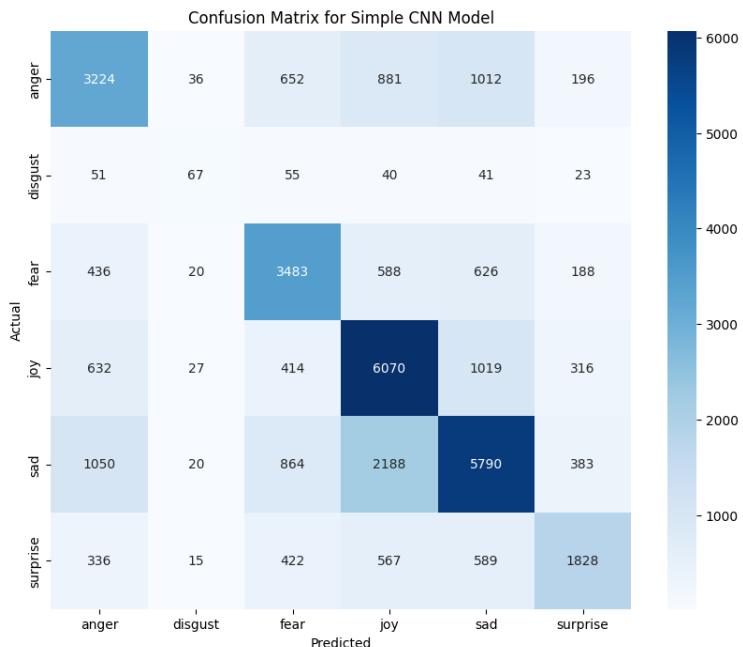
Simple CNN Model:

Accuracy: 0.5992

Precision: 0.5608

Recall: 0.5327

F1-score: 0.5416



شکل ۲۵ ماتریس آشتفتگی مدل CNN

## ارزیابی مدل LSTM

Simple LSTM Model:  
 Accuracy: 0.7884  
 Precision: 0.7753  
 Recall: 0.7563  
 F1-score: 0.7637



شکل ۲۶ ماتریس آشتفتگی مدل LSTM

همانطور که مشاهده می‌شود، داده‌ها نامتوازن هستند و کلاس Sad بیشترین پیش‌بینی درست را دارد. این موضوع بر پaramتر دقت تاثیر بیشتری دارد. پس نیاز است که معیارهای دیگری برای ارزیابی مثل F1 استفاده کنیم. در ادامه سه روش مختلف برای محاسبه میانگین معیارهای مذکور را بررسی می‌کنیم:

CNN-LSTM Model:  
 Macro Averaging - Precision: 0.7591, Recall: 0.7391, F1-score: 0.7483  
 Micro Averaging - Precision: 0.7691, Recall: 0.7691, F1-score: 0.7691  
 Weighted Averaging - Precision: 0.7702, Recall: 0.7691, F1-score: 0.7693

Simple CNN Model:  
 Macro Averaging - Precision: 0.5608, Recall: 0.5327, F1-score: 0.5416  
 Micro Averaging - Precision: 0.5992, Recall: 0.5992, F1-score: 0.5992  
 Weighted Averaging - Precision: 0.6010, Recall: 0.5992, F1-score: 0.5965

Simple LSTM Model:  
 Macro Averaging - Precision: 0.7753, Recall: 0.7563, F1-score: 0.7637  
 Micro Averaging - Precision: 0.7884, Recall: 0.7884, F1-score: 0.7884  
 Weighted Averaging - Precision: 0.7902, Recall: 0.7884, F1-score: 0.7883

## مقایسه روش‌های میانگین‌گیری

**Macro Averaging:** میانگین ساده معیارهای ارزیابی برای هر کلاس بدون توجه به تعداد نمونه‌های هر کلاس. این روش تمام کلاس‌ها را به یک اندازه در نظر می‌گیرد و برای ارزیابی عملکرد مدل در کلاس‌هایی با تعداد نمونه‌های کم مناسب است. در این دیتاست برخی کلاس‌ها تعداد نمونه‌های کمی دارند و macro averaging می‌تواند معیار تاثیرپذیری نسبت به این کلاس‌ها باشد.

**Micro Averaging:** میانگین معیارهای ارزیابی بر اساس کل نمونه‌ها، بدون توجه به کلاس. این روش برای ارزیابی کلی عملکرد مدل مناسب است و بیشتر تحت تاثیر کلاس‌هایی با تعداد نمونه‌های بیشتر قرار می‌گیرد.

**Weighted Averaging:** میانگین معیارهای ارزیابی بر اساس وزن هر کلاس که به تعداد نمونه‌های هر کلاس بستگی دارد. این روش ترکیبی از micro و macro averaging است و تاثیر تعداد نمونه‌های هر کلاس را در نظر می‌گیرد. این روش می‌تواند توازن خوبی بین دقت در کلاس‌هایی با تعداد نمونه بالا و کم ایجاد کند.

مدل LSTM ساده به طور کلی بهترین عملکرد را داشته و توانسته است دقت بالاتری نسبت به مدل‌های دیگر ارائه دهد.

جدول ۱ خروجی روش‌های یادگیری عمیق و یادگیری ماشین

Model	Accuracy	Precision	Recall	F1-score
CNN	0.5992	0.5608	0.5327	0.5416
LSTM	0.7884	0.7753	0.7563	0.7637
CNN-LSTM	0.7691	0.7591	0.7391	0.7483
Gaussian Naive Bayes	0.6804	0.6167	0.6972	0.6059
Decision Tree	0.8684	0.8781	0.8851	0.8814
Gradient Boosting	0.8886	0.9197	0.8964	0.9047
Random Forest	0.8935	0.9047	0.9039	0.9038
Logistic Regression	0.8902	0.9054	0.9011	0.9029

## ۱-۶. امتیازی - کیسه کلمات

در این بخش، از روش کیسه کلمات Bag of Words برای نمایش ویژگی‌های توییت‌ها استفاده می‌کنیم و مدل‌های ماشین لرنینگ که در مقاله ذکر شده‌اند را آموزش داده و ارزیابی می‌کنیم.

### Bag of Words

```
vectorizer = CountVectorizer(max_features=1000)
X_bow = vectorizer.fit_transform(data['cleaned_tweet']).toarray()
y = data['emotion'].factorize()[0]

print("Shape of Bag of Words features:", X_bow.shape)

Shape of Bag of Words features: (113829, 1000)
+ Code + Markdown

data_sampled, _, y_sampled, _ = train_test_split(X_bow, y, test_size=0.8, random_state=42)
X_train_bow, X_test_bow, y_train_bow, y_test_bow = train_test_split(data_sampled, y_sampled, test_size=0.2, random_state=42)

print("Training data shape:", X_train_bow.shape)
print("Test data shape:", X_test_bow.shape)

Training data shape: (18212, 1000)
Test data shape: (4553, 1000)
```

شکل ۲۷ روش کیسه کلمات

در این پیاده‌سازی فقط ۱۰۰۰ واژه برتر که بیشترین تکرار را دارند را در نظر می‌گیریم. برای افزایش سرعت آموزش نیز از ۲۰ درصد داده‌ها نمونه‌برداری می‌کنیم. خروجی هر کدام از روش‌ها به صورت زیر است:

Gaussian Naive Bayes Model:  
Accuracy: 0.6804  
Precision: 0.6167  
Recall: 0.6972  
F1-score: 0.6059

Decision Tree Model:  
Accuracy: 0.8684  
Precision: 0.8781  
Recall: 0.8851  
F1-score: 0.8814

Gradient Boosting Model:  
Accuracy: 0.8886  
Precision: 0.9197  
Recall: 0.8964  
F1-score: 0.9047

Random Forest Model:  
Accuracy: 0.8935

Precision: 0.9047  
Recall: 0.9039  
F1-score: 0.9038

Logistic Regression Model:  
Accuracy: 0.8902  
Precision: 0.9054  
Recall: 0.9011  
F1-score: 0.9029

## تحلیل نتایج

نتایج نشان می‌دهند که مدل‌های یادگیری ماشین مانند Gradient Boosting و Random Forest در روش کیسه کلمات، عملکرد بسیار خوبی دارند و حتی در برخی موارد بهتر از مدل‌های عمیق عمل می‌کنند. این موضوع می‌تواند به دلایل زیر باشد:

**садگی داده‌ها:** داده‌های توبیت‌های عمومی ممکن است ساختار ساده‌تری نسبت به توبیت‌های سیاسی داشته باشند و ویژگی‌های استخراج شده از روش کیسه کلمات بتواند اطلاعات کافی برای مدل‌های یادگیری ماشین فراهم کند.

**کمتربودن پارامترها:** مدل‌های ماشین لرنینگ پارامترهای کمتری نسبت به مدل‌های یادگیری عمیق دارند و به همین دلیل ممکن است در مجموعه داده‌های کوچکتر (۲۰ درصد داده‌ها) یا با ویژگی‌های ساده‌تر عملکرد بهتری داشته باشند.

**تنظیمات بهینه‌تر:** مدل‌های یادگیری ماشین با استفاده از روش‌هایی مانند کیسه کلمات و تنظیمات بهینه، می‌توانند به خوبی آموزش ببینند و نتایج قابل قبولی ارائه دهند. در صورتی که تنظیماتی که در این مسئله برای مدل‌های یادگیری عمیق بررسی کردیم مربوط به مقاله‌ای با دیتاست متفاوتی بود. پس ممکن است مدل‌هایی که پیاده‌سازی کرده بودیم، پارامترهایی بهتر از فضای جستجوی ارائه شده در این تمرین را داشته باشند و به دقت بالاتری برسند.

## پرسش ۲ - سامانه های سایبرفیزیکی: نگهداری هوشمند

### بخش ۱-۲: پیش پردازش داده ها:

- مجموعه داده مورد نظر توسط NASA برای بررسی شرایط موتور های توربوفن گردآوری شده است، این مجموعه داده عملاً داده صنعتی به حساب می آید، و هدف آن تخمین عمر باقی مانده قطعات صنعتی است، در ادامه به بررسی ستون های دیتا به تفکیک آموزش و تست میپردازیم:

○ داده آموزش:

شامل:

▪ که اندیس دستگاهی است که دیتا از آن تولید شده است. Unit number

▪ شماره cycle که دیتا در آن cycle توسط سنسورها اندازه گیری شده است. Time in cycle

▪ سه ستون operational setting: این ستون های، شرایط عملیاتی است که وضعیت دستگاه در حین آن اندازه گیری شده است.

▪ بیست و یک ستون sensor measurement: که شامل مقادیر اندازه گیری شده توسط ۲۱ سنسور در هر سایکل است.

ما از کتابخانه pandas برای خواندن و بررسی این دیتا استفاده کردیم که خروجی آن به شرح زیر است:

Unnamed: 0	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 12	sensor measurement 13	sensor measurement 14	sensor measurement 15	sensor measurement 16	sensor measurement 17	sensor measurement 18	
0	0	1	1	-0.0007	-0.0004	100.0	518.67	641.82	1589.70	1400.60	...	521.66	2388.02	8138.62	8.4195	0.03	392	2388
1	1	1	2	0.0019	-0.0003	100.0	518.67	642.15	1591.82	1403.14	...	522.28	2388.07	8131.49	8.4318	0.03	392	2388
2	2	1	3	-0.0043	0.0003	100.0	518.67	642.35	1587.99	1404.20	...	522.42	2388.03	8133.23	8.4178	0.03	390	2388
3	3	1	4	0.0007	0.0000	100.0	518.67	642.35	1582.79	1401.87	...	522.86	2388.08	8133.83	8.3682	0.03	392	2388
4	4	1	5	-0.0019	-0.0002	100.0	518.67	642.37	1582.85	1406.22	...	522.19	2388.04	8133.80	8.4294	0.03	393	2388

5 rows × 27 columns

شکل ۲۸ - داده های آموزشی

دقیقا همین دیتا برای داده تست نیز موجود می باشد، اما یک تفاوت اساسی در داده تست و آموزش موجود است:

تعداد سایکل های داده های آموزش تا جایی که قطعه از کار افتاده ادامه پیدا کرده است، یعنی آخرین سایکل عمر قطعه است.

در مورد داده های تست اینطور نیست، بلکه از جایی به بعد اندازه گیری قطع شده است اما در فایل دیگری، عمر قطعه بعد از آخرین سایکل اندازه گیری توسط سنسورها به نام RUL ذخیره شده است.

	Unnamed: 0	RUL	unit number
0	0	112	1
1	1	98	2
2	2	69	3
3	3	82	4
4	4	91	5

شکل ۲۹ - RUL داده های تست

- سپس به پیش پردازش داده ها اقدام کردیم:

#### Data selection ○

در این بخش مقاله به حذف برخی ستون هایی از دیتا که تاثیری در تخمین عمر قطعه ندارند میپردازد، این ستون ها را نیز به صورت آماری و بر اساس بررسی ۱۰ موتور انجام میدهد، با توجه به یکسان بودن صدرصدی دیتای مقاله با دیتای مورد استفاده ما، ما نیز همین ستون ها را حذف کردیم:

train_data.drop(['Unnamed: 0','operational setting 3', 'sensor measurement 1', 'sensor measurement 5', 'sensor measurement 10', 'sensor measurement 16','sensor measurement 19', 'sensor measurement 19'], axis=1, inplace=True)																																																																																																											
test_data.drop(['Unnamed: 0','operational setting 3', 'sensor measurement 1', 'sensor measurement 5', 'sensor measurement 10', 'sensor measurement 16','sensor measurement 19', 'sensor measurement 19'], axis=1, inplace=True)																																																																																																											
RUL_data.drop(['Unnamed: 0'], axis=1, inplace=True)																																																																																																											
train_data.head()																																																																																																											
<table border="1"> <thead> <tr> <th>unit number</th> <th>time in cycles</th> <th>operational setting 1</th> <th>operational setting 2</th> <th>sensor measurement 2</th> <th>sensor measurement 3</th> <th>sensor measurement 4</th> <th>sensor measurement 6</th> <th>sensor measurement 7</th> <th>sensor measurement 8</th> <th>sensor measurement 9</th> <th>sensor measurement 11</th> <th>sensor measurement 12</th> <th>sensor measurement 13</th> <th>sensor measurement 14</th> <th>sensor measurement 15</th> <th>sensor measurement 17</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>1</td> <td>1</td> <td>-0.0007</td> <td>-0.0004</td> <td>641.82</td> <td>1589.70</td> <td>1400.60</td> <td>21.61</td> <td>554.36</td> <td>2388.06</td> <td>9046.19</td> <td>47.47</td> <td>521.66</td> <td>2388.02</td> <td>8138.62</td> <td>8.4195</td> <td>392</td> </tr> <tr> <td>1</td> <td>1</td> <td>2</td> <td>0.0019</td> <td>-0.0003</td> <td>642.15</td> <td>1591.82</td> <td>1403.14</td> <td>21.61</td> <td>555.75</td> <td>2388.04</td> <td>9044.07</td> <td>47.49</td> <td>522.28</td> <td>2388.07</td> <td>8131.49</td> <td>8.4318</td> <td>392</td> </tr> <tr> <td>2</td> <td>1</td> <td>3</td> <td>-0.0043</td> <td>0.0003</td> <td>642.35</td> <td>1587.99</td> <td>1404.20</td> <td>21.61</td> <td>554.26</td> <td>2388.08</td> <td>9052.94</td> <td>47.27</td> <td>522.42</td> <td>2388.03</td> <td>8133.23</td> <td>8.4178</td> <td>390</td> </tr> <tr> <td>3</td> <td>1</td> <td>4</td> <td>0.0007</td> <td>0.0000</td> <td>642.35</td> <td>1582.79</td> <td>1401.87</td> <td>21.61</td> <td>554.45</td> <td>2388.11</td> <td>9049.48</td> <td>47.13</td> <td>522.86</td> <td>2388.08</td> <td>8133.83</td> <td>8.3682</td> <td>392</td> </tr> <tr> <td>4</td> <td>1</td> <td>5</td> <td>-0.0019</td> <td>-0.0002</td> <td>642.37</td> <td>1582.85</td> <td>1406.22</td> <td>21.61</td> <td>554.00</td> <td>2388.06</td> <td>9055.15</td> <td>47.28</td> <td>522.19</td> <td>2388.04</td> <td>8133.80</td> <td>8.4294</td> <td>393</td> </tr> </tbody> </table>	unit number	time in cycles	operational setting 1	operational setting 2	sensor measurement 2	sensor measurement 3	sensor measurement 4	sensor measurement 6	sensor measurement 7	sensor measurement 8	sensor measurement 9	sensor measurement 11	sensor measurement 12	sensor measurement 13	sensor measurement 14	sensor measurement 15	sensor measurement 17	0	1	1	-0.0007	-0.0004	641.82	1589.70	1400.60	21.61	554.36	2388.06	9046.19	47.47	521.66	2388.02	8138.62	8.4195	392	1	1	2	0.0019	-0.0003	642.15	1591.82	1403.14	21.61	555.75	2388.04	9044.07	47.49	522.28	2388.07	8131.49	8.4318	392	2	1	3	-0.0043	0.0003	642.35	1587.99	1404.20	21.61	554.26	2388.08	9052.94	47.27	522.42	2388.03	8133.23	8.4178	390	3	1	4	0.0007	0.0000	642.35	1582.79	1401.87	21.61	554.45	2388.11	9049.48	47.13	522.86	2388.08	8133.83	8.3682	392	4	1	5	-0.0019	-0.0002	642.37	1582.85	1406.22	21.61	554.00	2388.06	9055.15	47.28	522.19	2388.04	8133.80	8.4294	393
unit number	time in cycles	operational setting 1	operational setting 2	sensor measurement 2	sensor measurement 3	sensor measurement 4	sensor measurement 6	sensor measurement 7	sensor measurement 8	sensor measurement 9	sensor measurement 11	sensor measurement 12	sensor measurement 13	sensor measurement 14	sensor measurement 15	sensor measurement 17																																																																																											
0	1	1	-0.0007	-0.0004	641.82	1589.70	1400.60	21.61	554.36	2388.06	9046.19	47.47	521.66	2388.02	8138.62	8.4195	392																																																																																										
1	1	2	0.0019	-0.0003	642.15	1591.82	1403.14	21.61	555.75	2388.04	9044.07	47.49	522.28	2388.07	8131.49	8.4318	392																																																																																										
2	1	3	-0.0043	0.0003	642.35	1587.99	1404.20	21.61	554.26	2388.08	9052.94	47.27	522.42	2388.03	8133.23	8.4178	390																																																																																										
3	1	4	0.0007	0.0000	642.35	1582.79	1401.87	21.61	554.45	2388.11	9049.48	47.13	522.86	2388.08	8133.83	8.3682	392																																																																																										
4	1	5	-0.0019	-0.0002	642.37	1582.85	1406.22	21.61	554.00	2388.06	9055.15	47.28	522.19	2388.04	8133.80	8.4294	393																																																																																										

شکل ۳۰ - حذف ستون های بی استفاده در تخمین عمر

در این بخش به لیبل زدن دیتا پرداختیم که به تفکیک داده های آموزش

#### Data Labeling ○ و تست به توضیح آن میپردازیم:

داده های آموزش: گفتیم که آخرین سایکل داده های آموزش، همان عمر آن هاست چون بعد از آن عمل از کار افتاده اند، پس به سادگی به تفکیک هر موتور، آخرین سایکل کاری آن را استخراج کرده و با کسر کردن از سایر سایکل های آن موتور، تخمین عمر موتور در هر یک از سایکل های آن بدست می آید.

مقاله در این قسمت به مراجعی اشاره میکند که بهتر از حد بالای عمر را به ۱۳۰ سایکل محدود کنیم.

end-of-life. To better model the RUL changes along with time, in References [2, 7, 20], a piece-wise linear RUL target function was proposed, which limits the maximum RUL to a constant value and then starts linear degradation after a certain degree of usage. In this article, for C-MAPSS datasets, we set the maximum limit as 130 time cycles.

شکل ۳۱ - محدودیت حد بالای عمر به ۱۳۰ سایکل

همچنین برای طبقه بندی نیز، خیلی ساده از پنجره زمانی (بعدا توضیح داده خواهد شد) استفاده میکنیم اگر عمر وسیله از اندازه پنجره زمانی بیشتر باشد، آن را سالم(کلاس صفر) و اگر عمر وسیله در آن سایکل از پنجره زمانی کوچک تر باشد آن را خراب(کلاس یک) دسته بندی میکنیم.

مقاله عدد ۵۰ را به عنوان طول پنجره زمانی طبقه بندی معرفی میکند که دلیلش نیز گویاست، زیرا بعد از آموزش یک مدل اولیه، مشاهده میشود داده هایی که کمتر از ۵۰ سایکل از عمرشان باقی مانده دقت بالاتری در تخمین داشته اند.

showed that the hybrid CNN-LSTM regressive model gives more accurate forecasts when the value of the RUL is less than 50 cycles. Therefore, to label the data for classification, the time horizon  $w$  is set to 50 cycles, then the data will be labeled by two classes. The first class, noted class 0, represents the case where the engine RUL will be greater than the time horizon  $w$ , i.e.,  $RUL > 50$  (healthy condition). The second class, noted class 1, concerns the case where the engine RUL will not exceed the time horizon  $w$ , i.e.,  $RUL \leq 50$  (faulty condition).

شکل ۳۲ - طول پنجره برای طبقه بندی

در نتیجه ما نیز از همین روند پیروی کردیم که کد را در زیر میبینید:

```
train_data['RUI'] = 0
for i in range(1, 101):
    cycles = train_data[train_data['unit number']==i]['time in cycles']
    last_cycle = cycles.iloc[-1]
    RUI = last_cycle - cycles
    train_data.loc[train_data['unit number'] == i, 'RUI'] = RUI

train_data['RUI'] = np.where(train_data['RUI'].values>130, 130, train_data['RUI'])
cls_labels = np.where((train_data['RUI'].values>50, 0, 1))
train_data['Cls_labels'] = cls_labels
train_data.head()
```

شکل ۳۳ - کد بخش لیبل زدن

operational setting	2	sensor measurement	3	sensor measurement	4	sensor measurement	6	sensor measurement	7	sensor measurement	8	...	sensor measurement	12	sensor measurement	13	sensor measurement	14	sensor measurement	15	sensor measurement	17	sensor measurement	18	sensor measurement	20	sensor measurement	21	RUI	Cls_labels
-0.0004	641.82	1589.70	1400.60	21.61	554.36	2388.06	...	521.66	2388.02	8138.62	8.4195	392	2388	39.06	23.419	130	0													
-0.0003	642.15	1591.82	1403.14	21.61	553.75	2388.04	...	522.28	2388.07	8131.49	8.4318	392	2388	39.00	23.423	130	0													
0.0003	642.35	1587.99	1404.20	21.61	554.26	2388.08	...	522.42	2388.03	8133.23	8.4178	390	2388	38.95	23.344	130	0													
0.0000	642.35	1582.79	1401.87	21.61	554.45	2388.11	...	522.86	2388.08	8133.83	8.3662	392	2388	38.88	23.373	130	0													
-0.0002	642.37	1582.85	1406.22	21.61	554.00	2388.06	...	522.19	2388.04	8133.80	8.4294	393	2388	38.90	23.404	130	0													

شکل ۳۴ - ستون های اضافه شده بخش لیبل زدن

و در نهایت برای اطمینان درست از لیبل ها، تعداد دیتای هر کلاس را مقایسه کردیم:

```
train_data['Cls_labels'].value_counts()
```

```
Cls_labels
0    15531
1    5100
Name: count, dtype: int64
```

شکل ۳۵ - تعداد دیتای هر کلاس

داده های تست: در داده های تست همانطور که اشاره کردیم آخرین داده موجود زمان خرابی قطعه نیست، پس طبق راهنمایی ارزشمند موجود در صورت سوال، ابتدا آخر سایکل موجود از هر قطعه را با دیتای موجود در بخش فایل RUL جمع کرده و سپس از هر سطح داده ها کم کردیم که تخمین عمر قطعه بدست بیاید، همچنین محدود شده داده ها به ۱۳۰ سایکل در مسئله رگرسن و پنجره زمانی ۵۰ تایی برای طبقه بندی نیز اعمال شد.

```
test_data['RUI'] = 0
for i in range(1, 101):
    cycles = test_data[test_data['unit number']== i]['time in cycles']
    last_working_cycle = cycles.iloc[-1]
    last_cycle = RUL_data[RUL_data['unit number'] == i]['RUL'].values[0]
    RUI = last_cycle + last_working_cycle - cycles
    test_data.loc[test_data['unit number'] == i, 'RUI'] = RUI

test_data['RUI'] = np.where(test_data['RUI'].values>130, 130, test_data['RUI'])
cls_labels = np.where(test_data['RUI'].values>50, 0, 1)
test_data['Cls_labels'] = cls_labels
test_data.head()
```

شکل ۳۶ - استفاده از داده های RUL در بخش تست

در اینجا نیز برای اطمینان تعداد داده موجود در هر کلاس را بررسی نمودیم:

```
test_data['Cls_labels'].value_counts()
```

```
Cls_labels
0    12210
1     886
Name: count, dtype: int64
```

شکل ۳۷ - تعداد داده های هر کلاس در دیتای تست

از آنجایی که داده های سنسور های مختلف در یک scale نیستند، Data Normalization و مدل نیز نباید نسبت به مقادیر زیادتر یا کمتر بایاس شده و آن ها را بیش از اندازه مهم فرض کند و یا دست کم بگیرد، به نرمال کردن دیتا در بازه ۰ و ۱ میپردازیم، اما قبل از آن بخش تارگت و ویژگی ها را از هم جدا کرده و ستون های بی استفاده را نیز حذف میکنیم:

```
train_features = train_data.drop(['RUI', 'Cls_labels', 'unit number', 'time in cycles'], axis=1)
train_target = train_data[['RUI', 'Cls_labels']]

test_features = test_data.drop(['RUI', 'Cls_labels', 'unit number', 'time in cycles'], axis=1)
test_targets = test_data[['RUI', 'Cls_labels']]
```

+ Code + Markdown

```
scaler = MinMaxScaler()

train_features = scaler.fit_transform(train_features)
test_features = scaler.transform(test_features)
```

شکل ۳۸ - جداسازی و نرمال سازی ویژگی های تست و آموزش

در این بخش به پیاده سازی پنجره های زمانی پرداختیم به طوری که ابتدا اطلاعات هر یک از موتور ها جداسازی شد و سپس روی آن های تایم ویندوز لحاظ شد، به طوری که اندازه تایم ویندوز ها در این قسمت بنابر مقاله برابر با ۳۰ واحد در نظر گرفته شده زیرا کوتاه ترین دیتای آموزش، دارای ۳۱ سایکل آموزش بود، همچنین طبق خواسته مقاله Stride نیز برابر یک در نظر گرفته شد تا بیشتر تعداد دیتای ممکن تولید شود.

values of  $N_L$  large enough to benefit from the history information, but must be smaller than the minimum length of the recorded trajectories. Note that the minimum recorded length in C-MAPSS FD001 dataset is 31 cycles, therefore, the value of  $N_L$  is set to 30 cycles.

شکل ۳۹ - پنجره زمانی پیشنهادی مقاله

ing that short sliding strides can increase the number of training samples, which may reduce the risk of over-fitting, it is reasonable to set  $k = 1$ . Input pairs can be expressed

شکل ۴۰ - استراید پیشنهادی مقاله

```
window_size = 30
train_vectors = []
target_vectors = []
for i in range(1, 101):

    unit_data = train_features[train_features['unit number'] == i].values
    unit_target = train_target[train_target['unit number'] == i].values

    for j in range(len(unit_data) - window_size + 1):
        window_vec = unit_data[j:j+window_size, :-1]
        window_tar = unit_target[j:j+window_size, :-1]

        train_vectors.append(window_vec)
        target_vectors.append(window_tar)

train_vectors = np.array(train_vectors)
target_vectors = np.array(target_vectors)

train_target_reg = target_vectors[:, :, 0]
train_target_cls = target_vectors[:, :, 1]

window_train_target_reg = train_target_reg[:, :-1]
window_train_target_cls = train_target_cls[:, :-1]

print(train_target_reg.shape)
print(train_target_cls.shape)

print(train_vectors.shape)
print(window_train_target_reg.shape)
print(window_train_target_cls.shape)
```

شکل ۴۱ - کد ایجاد پنجره های زمانی

طبق پیشنهاد مقاله، از لیبل آخرین داده پنجره زمانی برای کل پنجره استفاده کردیم که در تصویر قرمز رنگ شده است.

در نهایت بردار های هر پنجره زمانی ابعاد  $30 \times 18$  داشته و به تعداد ۱۷۷۳۱ داده آموزشی خواهیم داشت، برای هر پنجره نیز دو لیبل یکی برای طبقه بندی و دیگری برای رگرسن خواهد بود.

(17731, 30, 18)  
(17731,)  
(17731,)

شکل ۴۲ - **shape** - بردارهای داده، و لیبل ها

با بررسی تعداد کلاس های مثبت و منفی در این قسمت نیز به اعداد زیر رسیدیم:

```
np.bincount(window_train_target_cls)
```

```
array([12631, 5100])
```

شکل ۴۳ - تعداد داده های هر کلاس در داده آموزش

این عملیات در دیتای تست نیز به همین شکل انجام شد و نتایج آن در نهایت به شکل زیر است:

```
(10196, 30, 18)  
(10196,)  
(10196,)
```

شکل ۴۴ - داده و لیبل ها در داده های تست

```
np.bincount(window_test_target_cls)
```

```
array([9310, 886])
```

شکل ۴۵ - تعداد داده در هر کلاس داده تست

در نهایت این بخش، به ساخت دیتاست با استفاده ازتابع دیتاست تنسورفلو پرداختیم، این کار ضروری نیست اما به سرعت بیشتر و بهینگی مدل بسیار کمک میکند:

```
train_dataset = tf.data.Dataset.from_tensor_slices((train_vectors, {'classification_output':window_train_target_cls, 'regression_output':window_train_target_reg}))  
train_dataset = train_dataset.shuffle(buffer_size=1000).batch(BATCH_SIZE).prefetch(buffer_size=tf.data.experimental.AUTOTUNE)  
  
test_dataset = tf.data.Dataset.from_tensor_slices((test_vectors, {'classification_output':window_test_target_cls, 'regression_output':window_test_target_reg}))  
test_dataset = test_dataset.batch(BATCH_SIZE).prefetch(buffer_size=tf.data.experimental.AUTOTUNE)
```

```
data, tar = next(iter(train_dataset))  
print(data.shape)  
print(tar['classification_output'].shape)  
print(tar['regression_output'].shape)  
  
(200, 30, 18)  
(200,)  
(200,)
```

شکل ۴۶ - ساخت دیتاست مورد نیاز از روی داده ها

## بخش ۲-۲: مدل سازی و ارزیابی:

الف) Classification

در این بخش به بررسی مدل CNN-LSTM Classifier میپردازیم:

```

input_shape = (train_vectors.shape[1], train_vectors.shape[2])
input_layer = Input(shape=input_shape)
x = Conv1D(32, 5, activation='relu')(input_layer)
x = Conv1D(64, 3, activation='relu')(x)
x = MaxPooling1D(pool_size=3)(x)
x = LSTM(50, return_sequences=True, dropout=0.2)(x)
x = Dropout(0.2)(x)
x = LSTM(50, dropout=0.2)(x)
x = Dropout(0.2)(x)
# x = Dense(512, activation='relu')(x)
classification_output = Dense(1, activation='sigmoid', name='classification_output')(x)
regression_output = Dense(1, activation='linear', name='regression_output')(x)
classifier_model = Model(inputs=input_layer, outputs=[classification_output, regression_output]) #outputs=[classification_output, regression_output])
classifier_model.summary()

```

شکل ۴۷ - معماری مدل CNN – LSTM

- بخش اول که لایه Input بر اساس سایز داده است.
- سپس دو لایه کانولوشن یک بعدی با تعداد فیلتر و اندازه مشخص شده استفاده شده است.
- سپس از یک MaxPooling استفاده کردیم.
- سپس دو لایه LSTM به همراه دو لایه Dropout استفاده کردیم.
- در نهایت یک بار با استفاده از لایه Dense پایانی و یک بار بدون استفاده از آن مدل را تعلیم دادیم، منظورم بخش قرمز است، در بخش طبقه بندهی به این لایه هیچ نیازی نبود و مدل به خوبی با پارامتر های موجود توانست توزیع دیتا را یادگیرد، مقاله نیز در این بخش واضح نیست، مقاله یکبار به استفاده از لایه Fully Connected اشاره میکند و در بخش پارامتر ها آن را لاحظ نمیکند:

Finally, fully connected layer is built on top of CNN-LSTM to process the outputs of LSTM, since it is good at mapping the learned feature representation to sample labels. In the output layer there is 1 neuron that will give us back the number of RUL and a linear activation.

The model weights are continuously updated by the backpropagation algorithm, as shown in

شکل ۴۹ - اشاره به Fully Connected نهایی

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	-

شکل ۴۸ - عدم ذکر لایه Fully Connected

ما نیز در این قسمت بنا بر آزمایش سعی کردیم مدل را اولا سنگین نکنیم تا overfit نشود، از طرفی اگر یادگیری واقع شد که نیازی به آن نباشد.

در نهایت نیز لایه Classification را لحاظ کردیم و ساختار مدل به شکل زیر شد:

Model: "functional\_1"

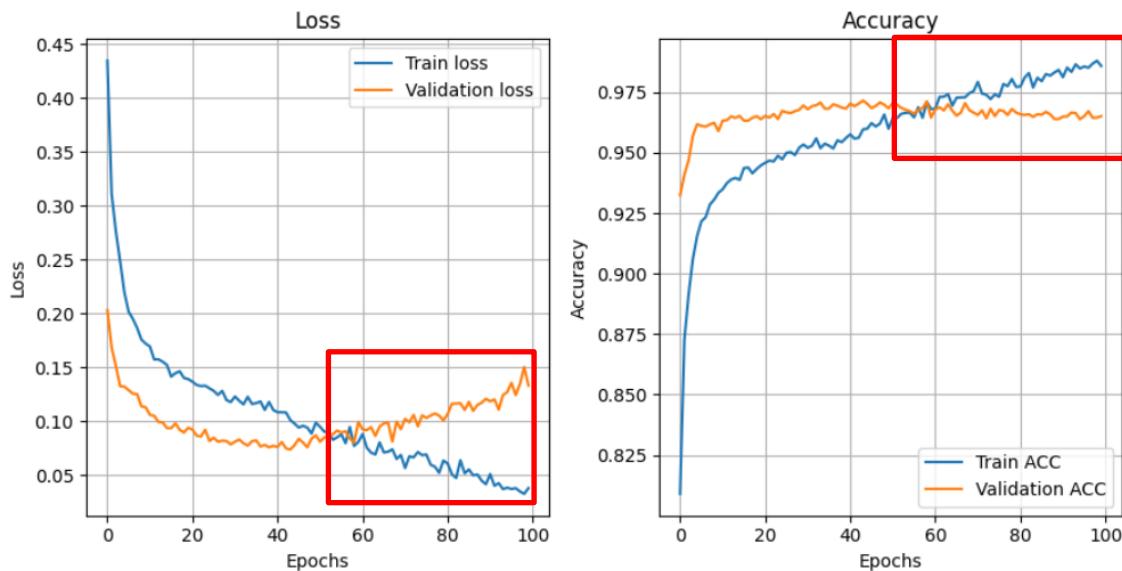
Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 30, 18)	0
conv1d (Conv1D)	(None, 26, 32)	2,912
conv1d_1 (Conv1D)	(None, 24, 64)	6,208
max_pooling1d (MaxPooling1D)	(None, 8, 64)	0
lstm (LSTM)	(None, 8, 50)	23,000
dropout (Dropout)	(None, 8, 50)	0
lstm_1 (LSTM)	(None, 50)	20,200
dropout_1 (Dropout)	(None, 50)	0
classification_output (Dense)	(None, 1)	51

Total params: 52,371 (204.57 KB)  
Trainable params: 52,371 (204.57 KB)  
Non-trainable params: 0 (0.00 B)

شکل ۵۰ - ساختار مدل CNN-LSTM CLS

سپس به آموزش مدل اقدام کردیم، در این قسمა از آپتمایزر adam بنا برخواسته سوال استفاده کردیم و نمودارهای آموزش به شکل زیر شد:

CNN-LSTM Classifier



شکل ۵۱ - نمودار آموزش مدل CNN - LSTM CLs

آنچه خودنمایی میکند، Overfitting در مدل است، زیرا هم Loss داده ولیدیشن بعد از مدتی افزایش یافته است و هم دقت بخش آموزش در حال افزایش بوده اما دقت validation ثابت شده است(بخش های قرمز رنگ)

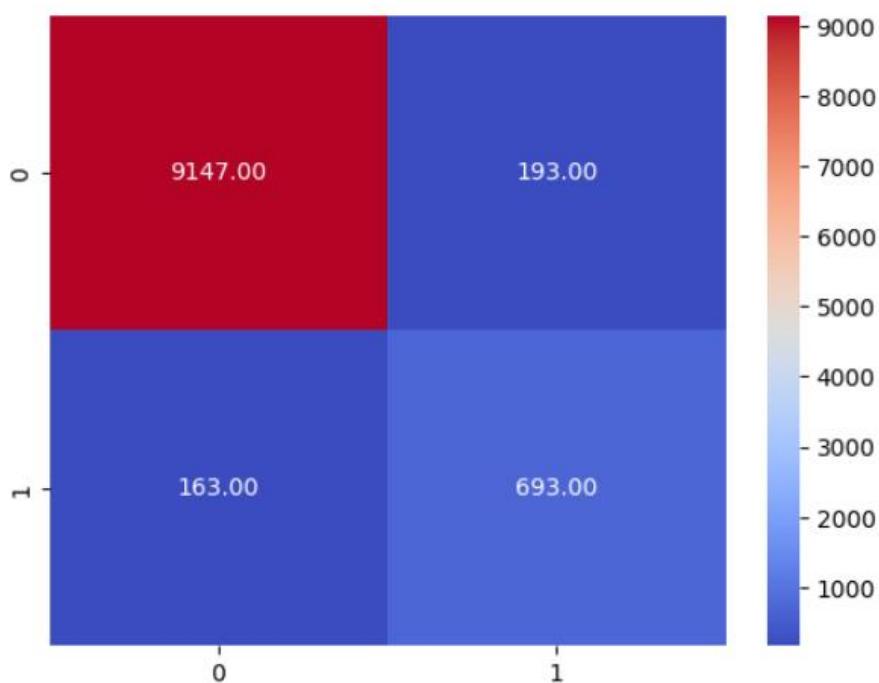
همچنین مدل را روی داده های تست Evaluate کردیم و مقادیر زیر را به عنوان Precision – Accuracy – F1-Score – Recall ثبت کردیم:

	1s 12ms/step			
	precision	recall	f1-score	support
0	0.98	0.98	0.98	9340
1	0.78	0.81	0.80	856
accuracy			0.97	10196
macro avg	0.88	0.89	0.89	10196
weighted avg	0.97	0.97	0.97	10196

شکل ۵۲ - نتایج مدل CNN - LSTM CLs

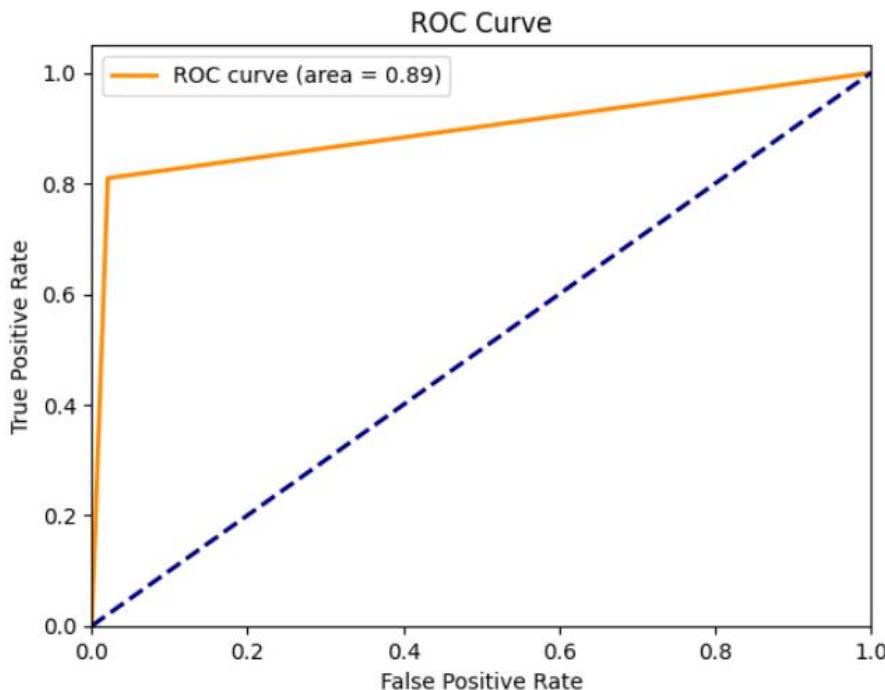
آنچه واضح است این است مدل در کلاس صفر که اکثر دیتای ما از آن کلاس است به خوبی عملکرده است و همه معیار های ارزیابی آن ۹۸ درصد است، و در کلاس ۱، دارای معیار های ارزیابی در حدود ۸۰ درصد است، این به دلیل بایاس بودن دیتا نسبت به یک کلاس قابل پیش بینی و درک است.

همچنین ماتریس درهم ریختگی مدل نیز به شکل زیر میباشد:



شکل ۵۳ - ماتریس درهم ریختگی مدل CNN - LSTM CLs

در ماتریس درهم ریختگی نیز نشان از همان چیزی است که در گزارش طبقه بندی قبل دیدیم و تعداد زیادی به صورت درست و تعداد کمتری به صورت اشتباه طبقه بندی شده اند و در نهایت ROC Curve مدل:



شکل ۵۴ - CNN-LSTM CLs مدل ROC

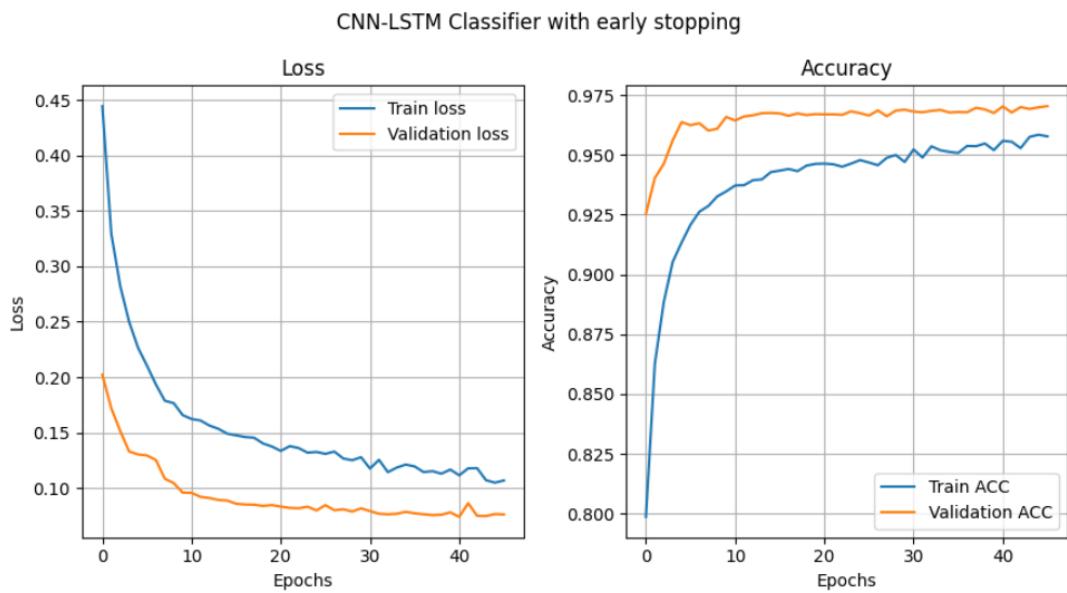
در این نمودار نیز میبینیم دقیق مدل و مساحت زیر نمودار مدل به حدود ۹۰ درصد رسیده است، اساساً این نمودار اگر به صورت ۹۰ درجه باشد، یعنی دقیق ۱۰۰ درصد مه اینجا نیز میبینیم همه چیز طبق انتظار قبلی مان است.

سپس در ادامه از **Early Stopping** در آموزش مدل استفاده کردیم و مشهود ترین مشاهده مان از بین رفتن در مدل و بهبود در نتایج مورد نظر بود:

```
early_stopping = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)

classifier_model.compile(
    optimizer='adam',
    loss={'classification_output': 'binary_crossentropy'},      # {'classification_output': 'binary_crossentropy',
    metrics={'classification_output': ['accuracy']} ,           # {'classification_output': ['accuracy'], 'regressi
)
classifier_history = classifier_model.fit(train_dataset,
    validation_data=test_dataset,
    batch_size=BATCH_SIZE,
    epochs=EPOCHS,
    callbacks=[early_stopping])
```

شکل ۵۵ - آموزش CNN – LSTM CLs با Early Stopping



شکل ۵۶ - نمودار های آموزش مدل با Early stoping

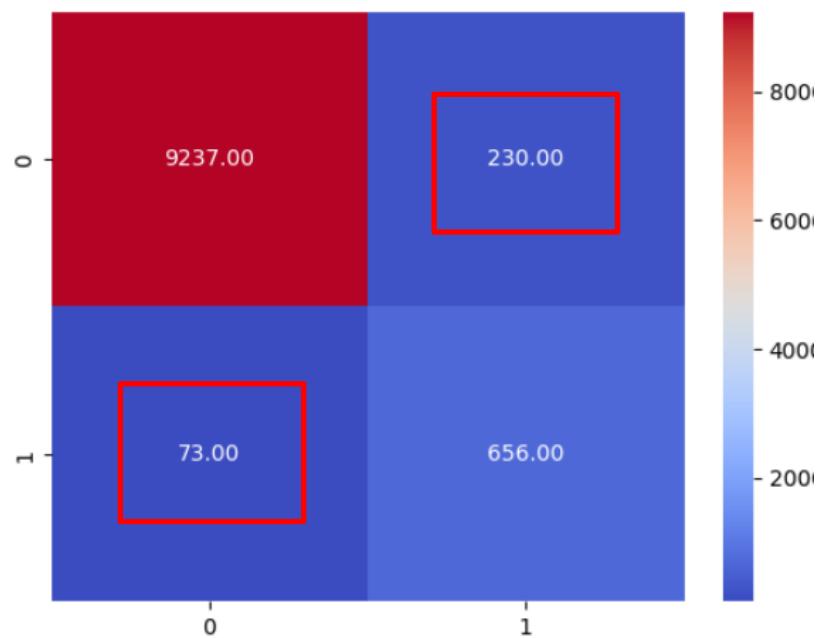
در این نمودار ها میبینیم که مدل دچار Overfitting نشده است.

همچنین طبق معیار های مورد نظر نیز، آنچه مشهود است، بهبود در Recall برای کلاس ۱ است و این بسیار مهم است، اگرچه Precision پایین آمده، این یعنی مدل سعی کرده به بهای اشتباہ گفتن بیشتر، تعداد بیشتری را کلاس ۱ پیش بینی کند تا Recall افزایش یابد و این در نهایت به نفع مدل است، زیرا بهتر از ما یک قطعه را سالم است، معیوب ارزیابی کنیم تا بر عکس، چرا که قطعه مورد نظر میتواند سبب سوانح جانی و مالی شود.

51/51		1s 12ms/step			
		precision	recall	f1-score	support
0	0.99	0.98	0.98	0.98	9467
1	0.74	0.90	0.81	0.81	729
accuracy				0.97	10196
macro avg		0.87	0.94	0.90	10196
weighted avg		0.97	0.97	0.97	10196

شکل ۵۷ - معیار های ارزیابی دقت مدل با Early Stoping

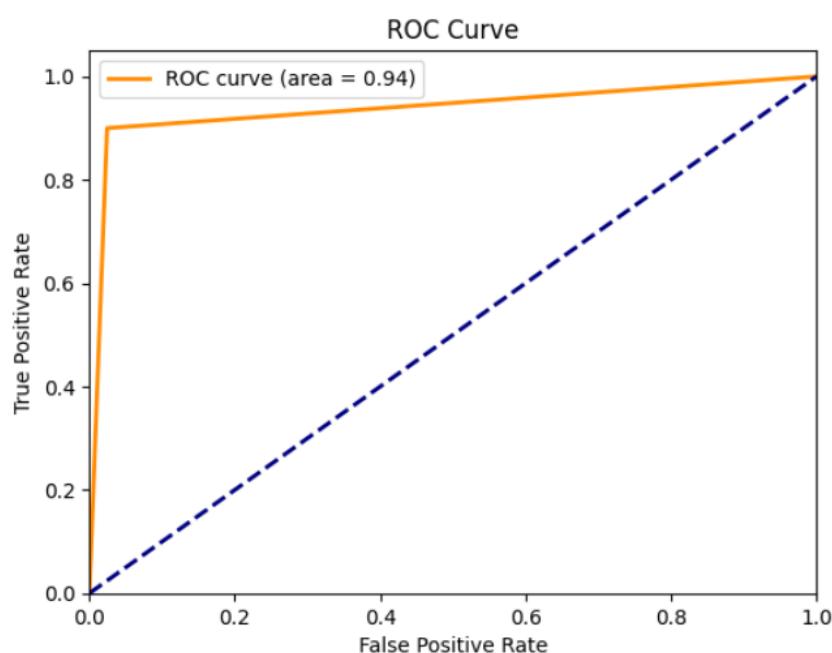
در ماتریس درهم ریختگی مدل نیز میبینیم:



شکل ۵۸ - ماتریس درهم ریختگی با Early Stopping

مدل به ازای افزایش اشتباه در کلاس صفر، توانسته اشتباهات کلاس یک را بسیار کم کند و این بسیار جالب است.

همین نتیجه در منحنی ROC نیز مشهود است که با افزایش حدود ۵ درصد در مقدار AUC رو به رو شده است.



شکل ۵۹ - ROC مدل با Early Stopping

## Regression -۲

در این بخش به پیاده سازی مدل Reg CNN-LSTM پرداختیم:

```
input_shape = (train_vectors.shape[1], train_vectors.shape[2])
input_layer = Input(shape=input_shape)
x = Conv1D(32, 5, activation='relu')(input_layer)
x = Conv1D(64, 3, activation='relu')(x)
x = MaxPooling1D(pool_size=3)(x)
x = LSTM(50, return_sequences=True, dropout=0.2)(x)
x = Dropout(0.2)(x)
x = LSTM(50, dropout=0.2)(x)
x = Dropout(0.2)(x)
x = Dense(512, activation='relu')(x)
# classification_output = Dense(1, activation='sigmoid', name='classification_output')(x)
regression_output = Dense(1, activation='linear', name='regression_output')(x)
regressor_model = Model(inputs=input_layer, outputs=regression_output) #outputs=[classification_output, regression_output])
regressor_model.summary()
```

شکل ۶۰ - کد ساختار مدل

تنها تفاوت با بخش قبلی اولا: در اضافه کردن یک لایه Dense به انتهای مدل قبل از بخش  
است و همچنین تغییر تابع فعالساز خروجی مدل از linear به sigmoid

افزوده شده لایه Dense به این دلیل است که مدل نیاز به پیچیدگی بیشتری دارد تا بتواند خروجی  
مورد نیاز را حدس بزند، چراکه مسئله ای که دارد اساسا مسئله سخت تری نسبت به طبقه بندی است،  
این موضوع را مقاله نیز اشاره میکند.

less data. The regressive network needs more data, although it provides **more information** about  
when the failure will happen. However, as noted in Section 1, training a single regressive network

شکل ۶۱ - سخت تر بودن شبکه Classification Regressive نسبت به

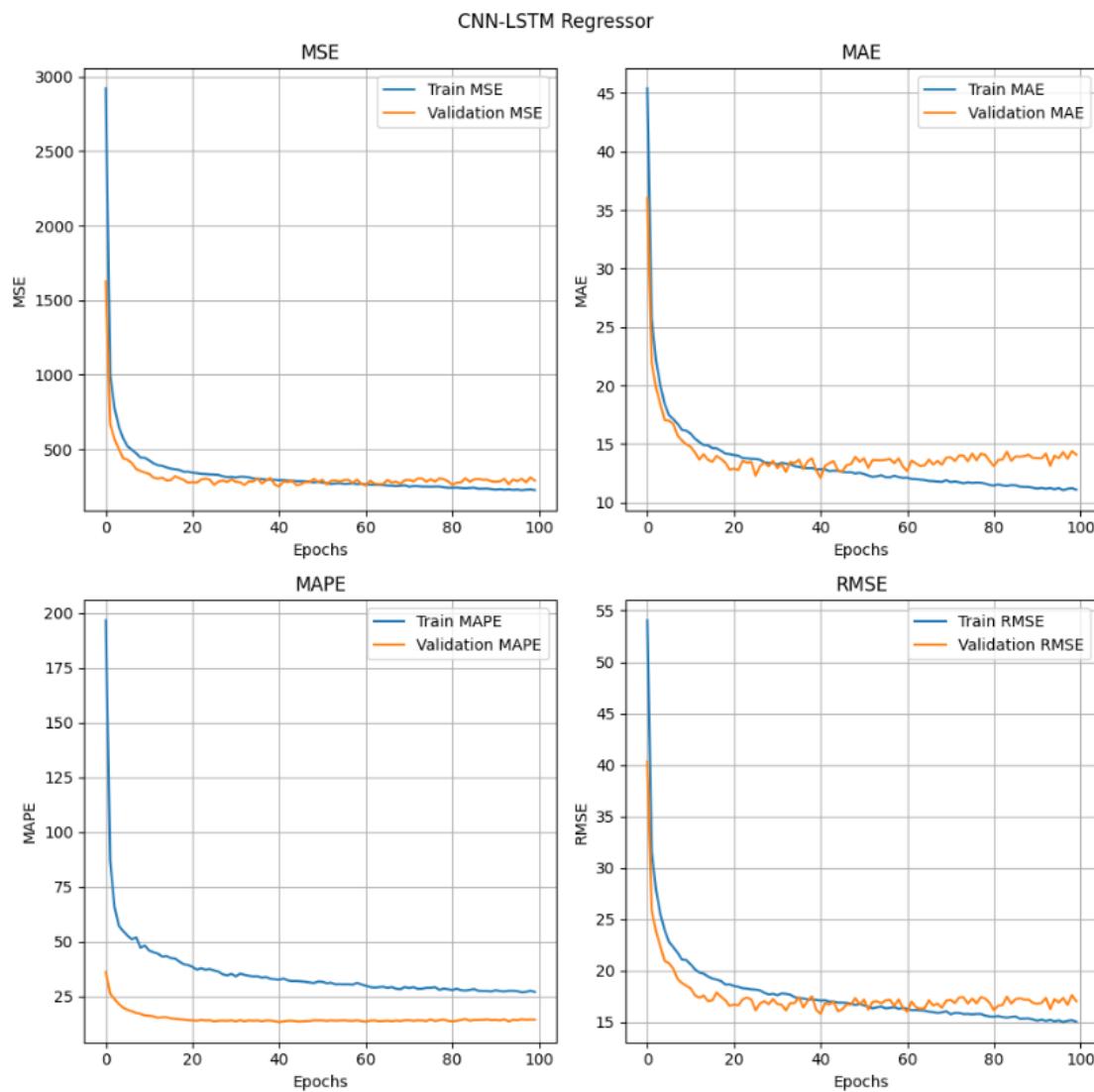
Model: "functional\_1"

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 30, 18)	0
conv1d (Conv1D)	(None, 26, 32)	2,912
conv1d_1 (Conv1D)	(None, 24, 64)	6,208
max_pooling1d (MaxPooling1D)	(None, 8, 64)	0
lstm (LSTM)	(None, 8, 50)	23,000
dropout (Dropout)	(None, 8, 50)	0
lstm_1 (LSTM)	(None, 50)	20,200
dropout_1 (Dropout)	(None, 50)	0
dense (Dense)	(None, 512)	26,112
regression_output (Dense)	(None, 1)	513

Total params: 78,945 (308.38 KB)  
Trainable params: 78,945 (308.38 KB)  
Non-trainable params: 0 (0.00 B)

شکل ۶۲ - ساختار شبکه Regressor

و در ادامه به آموزش این شبکه اقدام کردیم، در این قسمت از آپتیمایزر RMSprop استفاده کردیم.



شکل ۶۳ - نمودارهای آموزش شبکه در هر ۴ معیار خواسته شده

سپس داده تست را بر اساس همه پنجره‌ها و یک پنجره زمانی آخر داده‌های تست آزمودیم:

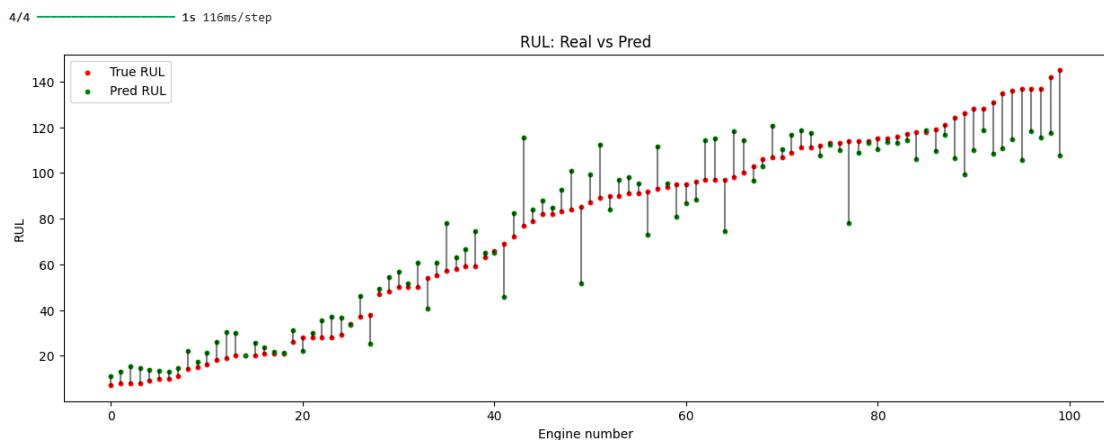
```
51/51 0s 8ms/step - loss: 262.2223 - mae: 13.5593 - mape: 13.1099 - rmse: 16.1816
all_window MSE: 290.92
all_window MAE: 14.09
all_window MAPE: 14.22
all_window RMSE: 17.06
```

شکل ۶۴ - معیارها روی همه پنجره‌های داده تست

```
4/4 1s 25ms/step - loss: 180.0932 - mae: 10.2703 - mape: 17.9111 - rmse: 13.4158
last_window_per_test_data MSE: 184.63
last_window_per_test_data MAE: 10.28
last_window_per_test_data MAPE: 18.68
last_window_per_test_data RMSE: 13.59
```

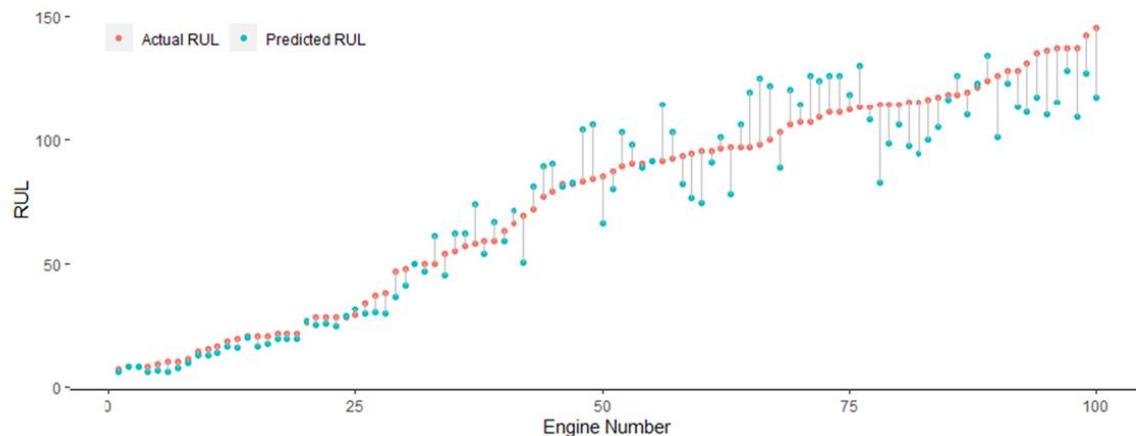
شکل ۶۵ - معیارها روی آخرین پنجره زمانی هر داده تست

سپس داده های تست را روی یک نمودار به ترتیب RUL صعودی رسم کردیم:



شکل ۶۶ - قدرت مدل در پیش بینی مقدار RUL هر داده

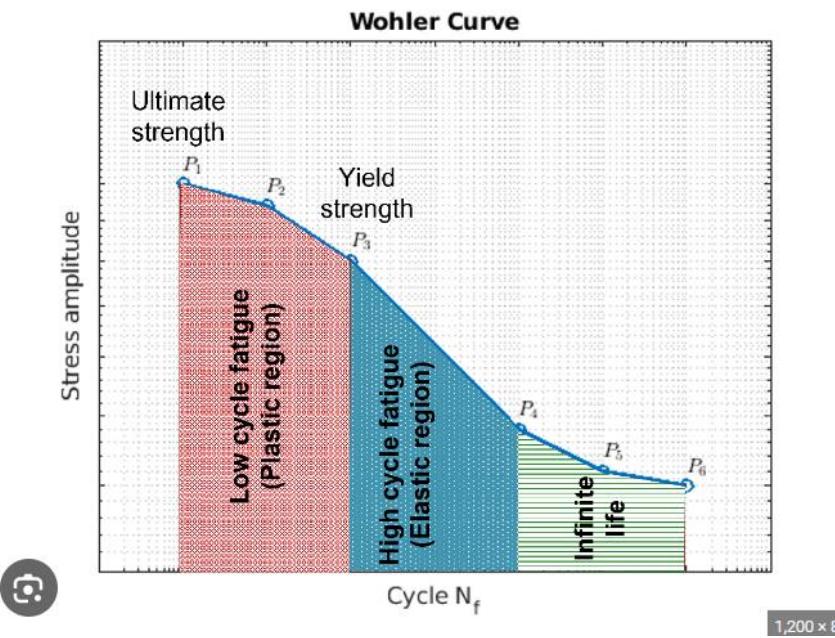
میبینیم که قدرت مدل در دیتا های دارای RUL کم بسیار خوب است و دقیق بالاتر و خطای کمتری دارد، اما هر چه RUL واقعی بزرگتر باشد، مدل نیز ضعف بیشتری از خود نشان میدهد، این موضوع با نتیجه مقاله نیز سازگار است:



شکل ۶۷ - تصویر از مقاله برای مقایسه با نتایج ما

البته این امر نیز واضح است، اولاً ما حد بالای تخمین ها را به ۱۳۰ سایکل محدود کرده ایم، در صورتی که لزوماً دیتای واقعی کمتر از ۱۳۰ سایکل نخواهد بود، و به همین دلیل اکثر تخمین های سمت راست نمودار، دارای خطای منفی شده اند(تخمین کمتر از حد واقعی)

دوماً وقتی تعداد سایکل زیادی تا خرایی موتور باقی مانده است، لزوماً سنسور ها چیز متفاوتی را حس نمیکنند، از نظر mechanical اگر بررسی کنیم، بسیار از collapse های مکانیکی، ناگهان اتفاق می افتد، در مکانیک موضوعی به اسم خستگی وجود دارد که تصویر زیر گویای آن است:



شکل - ۶۸ در مواد مکانیکال Fatigue

میبینیم که در سایکل های اولیه اثر محدودی وجود دارد و به ناگاه ماده collapse میکند.

سپس مدل را در حالتی که دارای Early Stopping باشد بررسی کردیم:

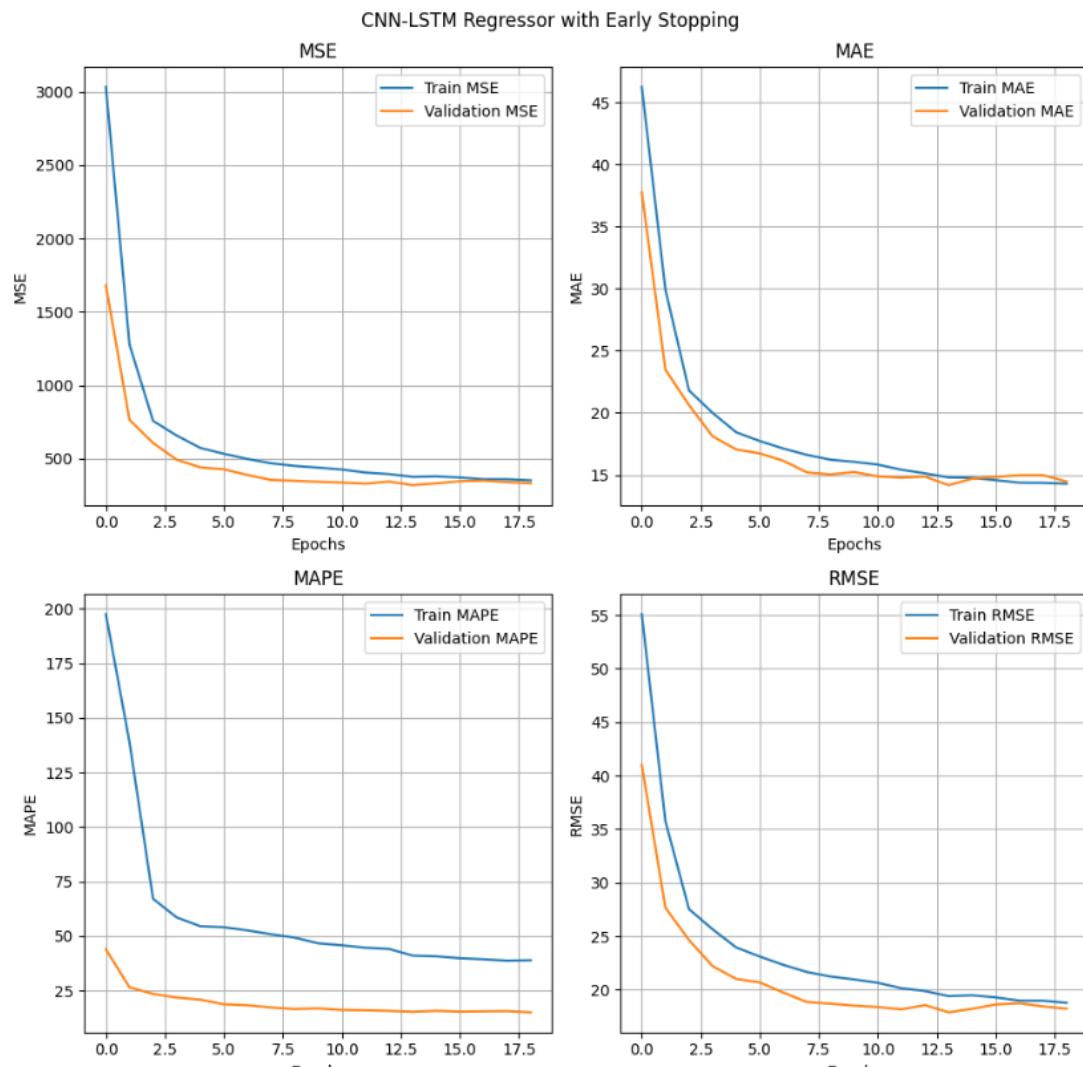
در این بخش که ساختار مدل یکسان است و تنها از Early stopping استفاده کردیم که مدل بعد از epoch ۱۹ متوقف شد.

```
early_stopping = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)

# tf.keras.backend.set_epsilon(1)
rmse = tf.keras.metrics.RootMeanSquaredError(name='rmse')

regressor_model.compile(
    optimizer='RMSprop',
    loss= {'regression_output': 'mean_squared_error'},      #{'classification_output': 'binary_crossentropy', 'r
    metrics={'regression_output': ['mae', 'mape', rmse]},      #{'classification_output': ['accuracy'], '
)
regressor_history = regressor_model.fit(train_dataset,
    validation_data=test_dataset,
    batch_size=BATCH_SIZE,
    epochs=EPOCHS,
    callbacks=[early_stopping])
```

نمودارهای مدل عبارتند از:

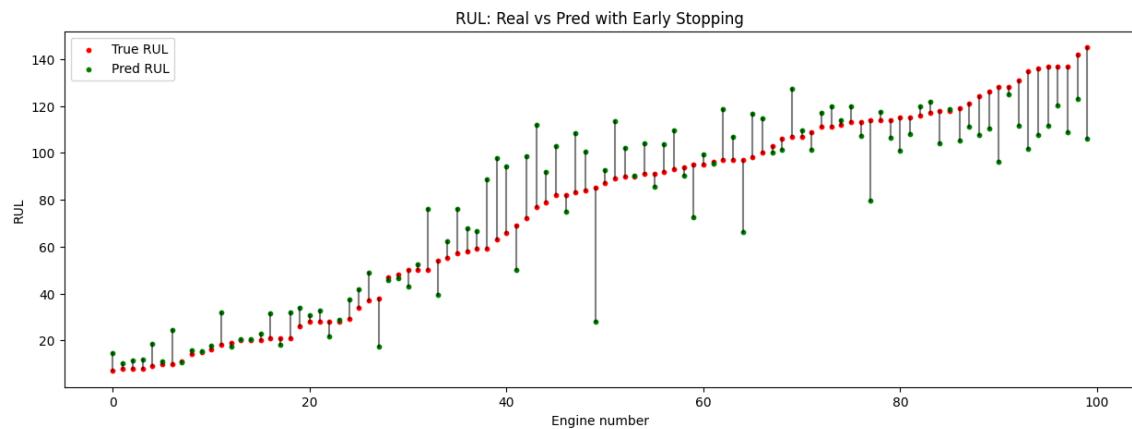


شکل ۶۹ - نمودارهای آموزش مدل با early stopping Regressor

با بررسی متريک ها روی همه پنجره های زمان و آخرین پنجره زمانی:

```
51/51 0s 7ms/step - loss: 274.8380 - mae: 13.2721 - mape: 13.8301 - rmse: 16.5521
all_window MSE: 320.24
all_window MAE: 14.18
all_window MAPE: 15.32
all_window RMSE: 17.90
#####
4/4 0s 8ms/step - loss: 253.1205 - mae: 12.0391 - mape: 20.0939 - rmse: 15.8904
last_window_per_test_data MSE: 273.90
last_window_per_test_data MAE: 12.45
last_window_per_test_data MAPE: 21.59
last_window_per_test_data RMSE: 16.55
```

میبینیم که خطای مدل روی این مسائل بیشتر از حالت قبل است که بدیهی است چرا که مدل تقریباً ۲۰ درصد زمان قبلی را آموزش دیده است اما به نگاه نمودار مقایسه مقادیر واقعی و تخمینی:



شکل ۷۰ - مقایسه مقادیر واقعی و تخمینی در مدل

در این بخش میبینیم موتور های دارای RUL کم، بسیار دقیق پیش بینی شده اند، نکته مهم همین است که وقتی موتوری در استانه از کار افتادگی است بتوانیم آن را دقیق تر شناسایی کنیم.

در نهایت دقیقیت مدل های با Early stopping و بدون آن در مدل CNN-Regressor و Classification به شکل زیر شد:

CNN – LSTM Classifier With Early Stopping		CNN – LSTM Classifier Without Early Stopping		
Class ۰	Class ۱		Class ۰	Class ۱
۰.۹۹	۰.۷۴	Precision	۰.۹۸	۰.۷۸
۰.۹۸	۰.۹۰	Recall	۰.۹۸	۰.۸۱
۰.۹۸	۰.۸۱	F1-Score	۰.۹۸	۰.۸۰
۰.۹۷		Accuracy		۰.۹۷

جدول ۲ - مقایسه مدل CNN-LSTM Classifier

CNN – LSTM Regressor With Early Stopping		CNN – LSTM Regressor Without Early Stopping		
All-windows	Last-windows		All-windows	Last-windows
۲۹۰.۹۲	۱۸۴.۶۳	MSE	۳۲۰.۲۴	۲۷۳.۹۰
۱۴.۰۹	۱۰.۲۸	MAE	۱۴.۱۸	۱۲.۴۵
۱۴.۲۲	۱۸.۶۸	MAPE	۱۵.۳۲	۲۱.۵۹
۱۷.۰۶	۱۳.۵۹	RMSE	۱۷.۹۰	۱۶.۵۵

جدول ۳ - مقایسه مدل های CNN-LSTM Regressor

### بخش ۲-۳: مقایسه با مدل های پایه:

در این بخش به توضیح مدل های CNN و LSTM اشاره شده در مقاله میپردازیم:

#### CNN – Classifier مدل

معماری مدل: که دقیقاً طبق مقاله شامل تنها دو لایه کانولوشنی است که در ادامه Flatt شده و یه لایه Dense روی آن قرار گرفته است.

```
input_shape = (train_vectors.shape[1], train_vectors.shape[2])
input_layer = Input(shape=input_shape)
x = Conv1D(32, 5, activation='relu')(input_layer)
x = Conv1D(64, 3, activation='relu')(x)
x = MaxPooling1D(pool_size=3)(x)
x = Flatten()(x)
x = Dense(128, activation='relu')(x)
classification_output = Dense(1, activation='sigmoid', name='classification_output')(x)
# regression_output = Dense(1, activation='linear', name='regression_output')(x)
cnn_classifier_model = Model(inputs=input_layer, outputs=classification_output) #outputs=[classification_output, regression_output])
cnn_classifier_model.summary()
```

شكل ۷۱-معماری CNN - Cls

ساختار مدل: که از نظر تعداد پارامتر تقریباً با مدل های قبلی برابر است و قابلیت مقایسه را ممکن میکند.

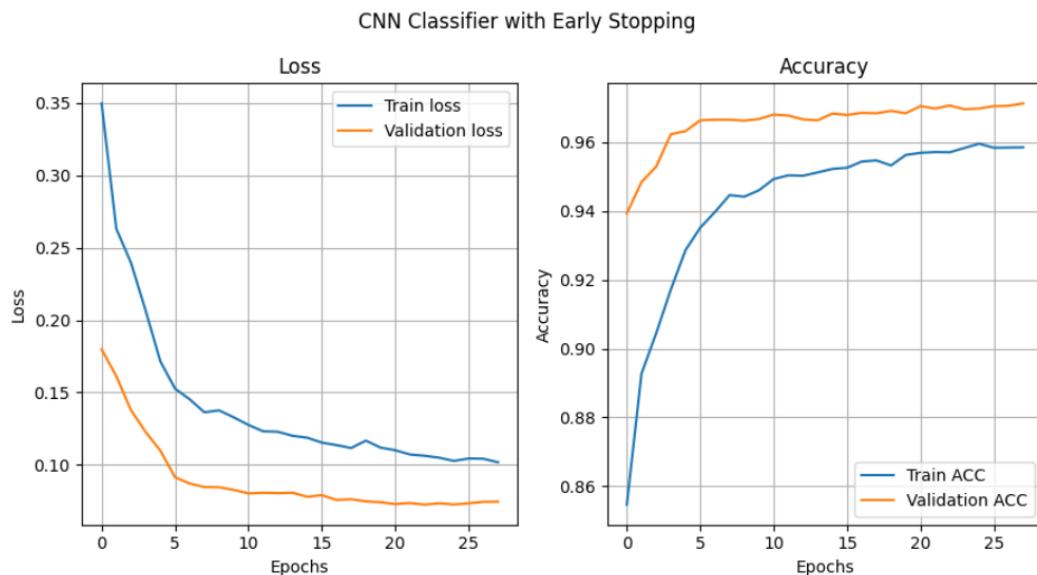
Model: "functional\_5"

Layer (type)	Output Shape	Param #
input_layer_2 (InputLayer)	(None, 30, 18)	0
conv1d_4 (Conv1D)	(None, 26, 32)	2,912
conv1d_5 (Conv1D)	(None, 24, 64)	6,208
max_pooling1d_2 (MaxPooling1D)	(None, 8, 64)	0
flatten (Flatten)	(None, 512)	0
dense_2 (Dense)	(None, 128)	65,664
classification_output (Dense)	(None, 1)	129

```
Total params: 74,913 (292.63 KB)
Trainable params: 74,913 (292.63 KB)
Non-trainable params: 0 (0.00 B)
```

شکل ۷۲ - ساختار مدل CNN - CLs

نتایج:

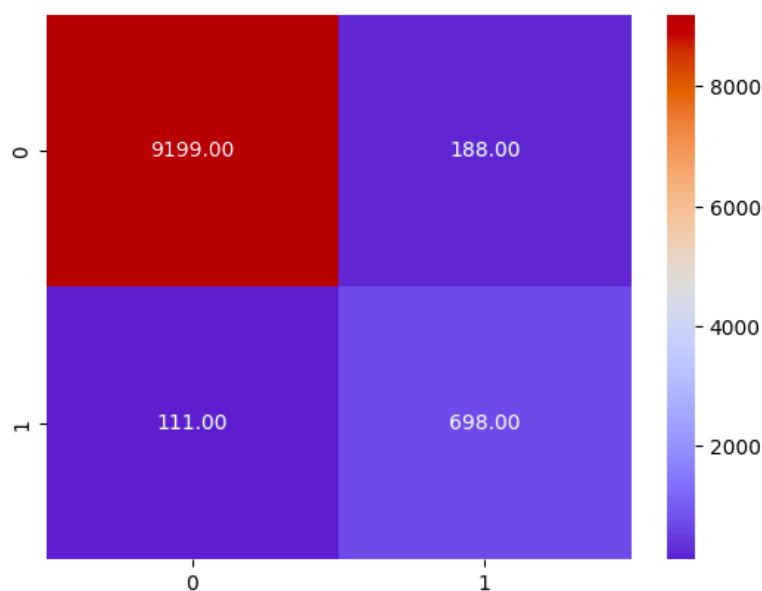


شکل ۷۳ - نمودار های Loss و ACC روی CNN - CLs

در نمودارهای بالا، شاهد آموزش خوب مدل هستیم، اگرچه دقت و تابع هزینه به اندازه مدل ترکیبی قبلی مناسب نیستند، اما این موضوع به وضوح قابل مشاهده است.

51/51	1s	7ms/step
	precision	recall
0	0.99	0.98
1	0.79	0.86
accuracy		0.97
macro avg	0.89	0.92
weighted avg	0.97	0.97

شکل ۷۴ - نتایج مدل CLs CNN



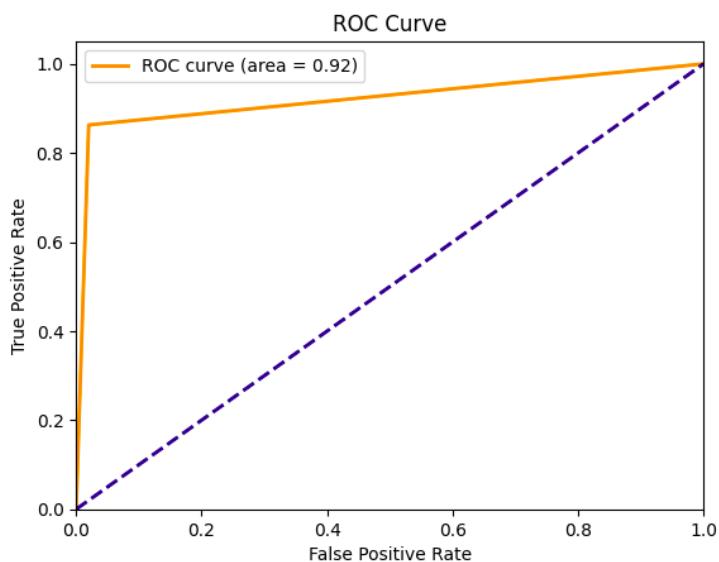
شکل ۷۵ - ماتریس پراکندگی مدل CNN – CLs

این مدل توانسته در مقایسه با مدل CNN – LSTM Classifier نتایج معقولی کسب کند، در recall مدل شاهد عملکرد قابل قبول کسب کند اگرچه به اندازه مدل CNN – LSTM نتوانسته موفق باشد، جدول زیر به مقایسه این دو مدل میپردازد.

مقایسه مدل:

CNN – LSTM Classifier With Early Stopping			CNN Classifier With Early Stopping	
Class ۰	Class ۱		Class ۰	Class ۱
۰.۹۹	۰.۷۴	Precision	۰.۹۹	۰.۰.۷۹
۰.۹۸	۰.۹۰	Recall	۰.۹۸	۰.۸۶
۰.۹۸	۰.۸۱	F <sup>۱</sup> -Score	۰.۹۸	۰.۸۲
۰.۹۷		Accuracy		۰.۹۷

جدول ۴ - مقایسه مدل CNN CLs , CNN-LSTM CLs



شکل ۷۶ - منحنی ROC

همچنین در مقایسه AUC این نمودار با نمودار قبلی نیز شاهد کاهش ۲ درصدی سطح زیر نمودار هستیم، از مقدار ۰.۹۴ به ۰.۹۲ رسیده است، بدیهی است که این مدل از مدل ترکیبی ضعیف تر است.

## :CNN – Regressor مدل

معماری مدل: که دقیقا طبق مقاله شامل تنها دو لایه کانولوشنی است که در ادامه Flatt شده و یه لایه Dense روی آن قرار گرفته است.

```
input_shape = (train_vectors.shape[1], train_vectors.shape[2])
input_layer = Input(shape=input_shape)
x = Conv1D(32, 5, activation='relu')(input_layer)
x = Conv1D(64, 3, activation='relu')(x)
x = MaxPooling1D(pool_size=3)(x)
x = Flatten()(x)
x = Dense(128, activation='relu')(x)
# classification_output = Dense(1, activation='sigmoid', name='classification_output')(x)
regression_output = Dense(1, activation='linear', name='regression_output')(x)
cnn_regressor_model = Model(inputs=input_layer, outputs=regression_output) #outputs=[classification_output, regression_output])
cnn_regressor_model.summary()
```

شکل ۷۷ - مدل CNN - Regressor

ساختار مدل: که از نظر تعداد پارامتر تقریبا با مدل های قبلی برابر است و قابلیت مقایسه را ممکن میکند.

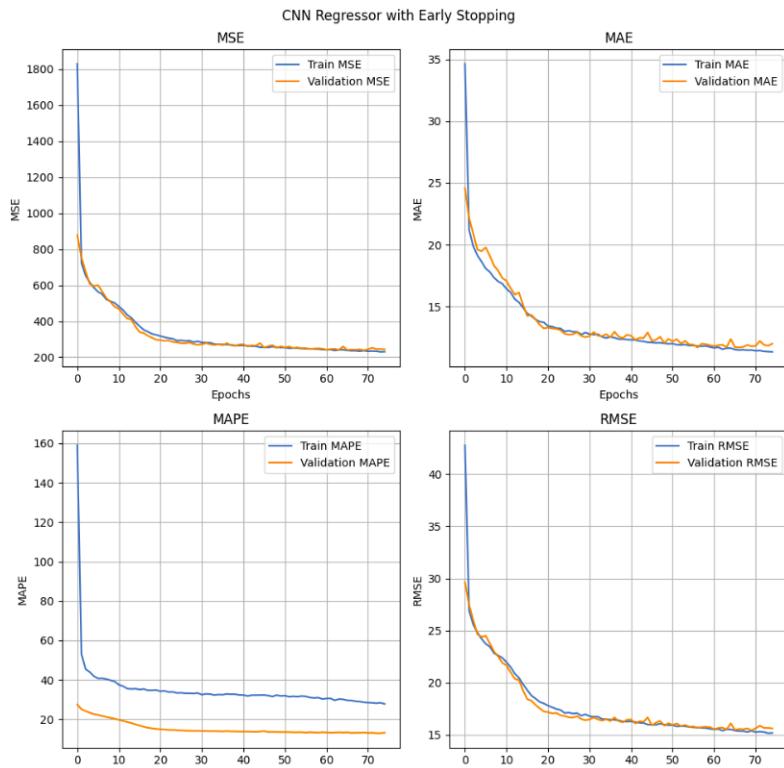
Model: "functional\_7"

Layer (type)	Output Shape	Param #
input_layer_3 (InputLayer)	(None, 30, 18)	0
conv1d_6 (Conv1D)	(None, 26, 32)	2,912
conv1d_7 (Conv1D)	(None, 24, 64)	6,208
max_pooling1d_3 (MaxPooling1D)	(None, 8, 64)	0
flatten_1 (Flatten)	(None, 512)	0
dense_3 (Dense)	(None, 128)	65,664
regression_output (Dense)	(None, 1)	129

```
Total params: 74,913 (292.63 KB)
Trainable params: 74,913 (292.63 KB)
Non-trainable params: 0 (0.00 B)
```

شکل ۷۸ - ساختار مدل CNN - Regressor

نتایج:



شکل ۷۹ - نمودار های ارزیابی CNN - Regressor

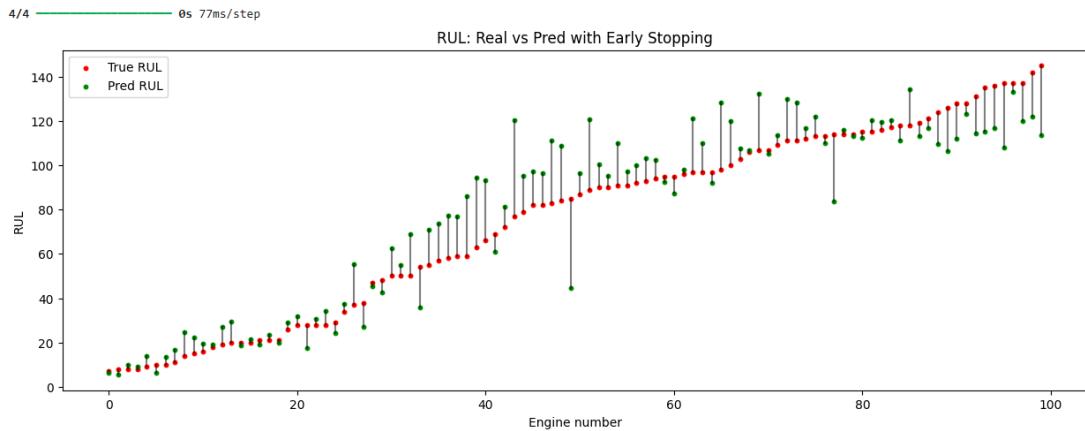
در نمودارهای بالا، شاهد آموزش خوب مدل هستیم، اگرچه دقت و تابع هزینه به اندازه مدل ترکیبی قبلی مناسب نیستند، اما این موضوع به وضوح قابل مشاهده است.

```
51/51 ----- 0s 2ms/step - loss: 211.5788 - mae: 11.0905 - mape: 11.9170 - rmse: 14.5151
all_window MSE: 238.14
all_window MAE: 11.78
all_window MAPE: 13.23
all_window RMSE: 15.43

#####
4/4 ----- 1s 133ms/step - loss: 214.3869 - mae: 11.2268 - mape: 17.6752 - rmse: 14.6274
last_window_per_test_data MSE: 227.86
last_window_per_test_data MAE: 11.50
last_window_per_test_data MAPE: 18.98
last_window_per_test_data RMSE: 15.09

#####
```

شکل ۸۰ - نتایج مدل CNN Regressor



شکل ۸۱ - مقایسه دیتای واقعی و پیش‌بینی توسط مدل CNN - Regressor

مقایسه مدل:

CNN – LSTM Regressor With Early Stopping			CNN Regressor With Early Stopping	
All-windows	Last-windows		All-windows	Last-windows
۲۹۰.۹۲	۱۸۴.۶۳	MSE	۲۳۸.۱۴	۲۲۷.۷۶
۱۴.۰۹	۱۰.۲۸	MAE	۱۱.۷۸	۱۱.۵۰
۱۴.۲۲	۱۸.۶۸	MAPE	۱۳.۲۳	۱۸.۹۸
۱۷.۰۶	۱۳.۵۹	RMSE	۱۵.۴۳	۱۵.۰۹

جدول ۵- مقایسه مدل CNN CLs , CNN-LSTM CLs

با مقایسه معیار ها، در حالت Last windows به ازای هر داده تست، که در مقاله نیز همین دیتا مورد بررسی قرار گرفته است، میبینیم مدل پیشنهادی مقاله نسبت به این مدل عملکرد بهتری داشته است، این مورد نیز بدیهی است زیرا اساسا مدل LSTM برای یادگیری داده های سری زمانی است، و این موضوع نیز در بخش قابل بررسی است.

## LSTM – Classifier مدل

معماری مدل که دقیقا طبق مقاله شامل تنها دو لایه LSTM است که در ادامه یک لایه Dense روی آن قرار گرفته است.

```
input_shape = (train_vectors.shape[1], train_vectors.shape[2])
input_layer = Input(shape=input_shape)
x = LSTM(50, return_sequences=True, dropout=0.2)(input_layer)
x = Dropout(0.2)(x)
x = LSTM(50, dropout=0.2)(x)
x = Dropout(0.2)(x)
x = Dense(128, activation='relu')(x)
classification_output = Dense(1, activation='sigmoid', name='classification_output')(x)
# regression_output = Dense(1, activation='linear', name='regression_output')(x)
lstm_classifier_model = Model(inputs=input_layer, outputs=[classification_output, regression_output]) #outputs=[classification_output, regression_output])
lstm_classifier_model.summary()
```

شکل ۸۲ - ساختار مدل LSTM - CLs

ساختار مدل: که از نظر تعداد پارامتر تقریبا با مدل های قبلی برابر است و قابلیت مقایسه را ممکن میکند.

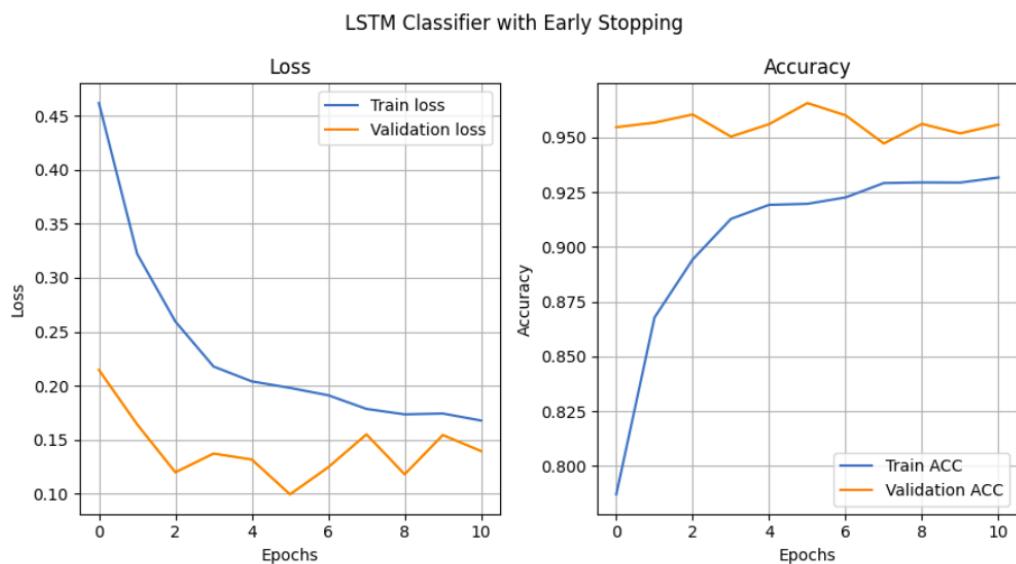
Model: "functional\_9"

Layer (type)	Output Shape	Param #
input_layer_4 (InputLayer)	(None, 30, 18)	0
lstm_4 (LSTM)	(None, 30, 50)	13,800
dropout_4 (Dropout)	(None, 30, 50)	0
lstm_5 (LSTM)	(None, 50)	20,200
dropout_5 (Dropout)	(None, 50)	0
dense_4 (Dense)	(None, 128)	6,528
classification_output (Dense)	(None, 1)	129

```
Total params: 40,657 (158.82 KB)
Trainable params: 40,657 (158.82 KB)
Non-trainable params: 0 (0.00 B)
```

شکل ۸۳ - ساختار مدل CNN – CLs

نتایج:

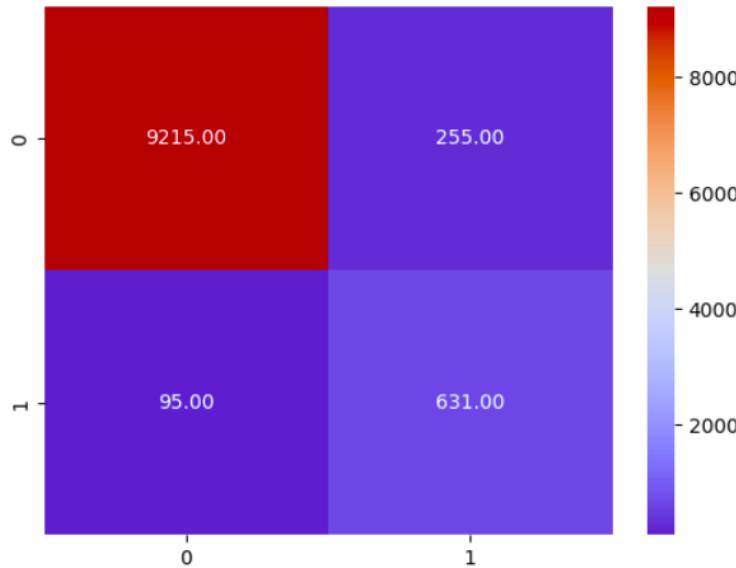


شکل ۸۴ - نمودارهای Loss و روی CLs

در نمودارهای بالا، به وضوح شاهد ضعف بیشتر مدل LSTM نسبت به CNN و همچنین نسبت به CNN – LSTM هستیم.

51/51		1s 22ms/step			
		precision	recall	f1-score	support
	0	0.99	0.97	0.98	9470
	1	0.71	0.87	0.78	726
		accuracy		0.97	10196
		macro avg		0.85	0.92
		weighted avg		0.97	0.97
				0.97	10196

شکل ۸۵ - نتایج مدل CNN CLs



**LSTM – Cls** شکل ۸۶ - ماتریس پراکندگی مدل

این مدل نیز در مقایسه با مدل اصلی، نتایج ضعیف تری دارد، آنچه به نظرم مهم است، دو مورد است:

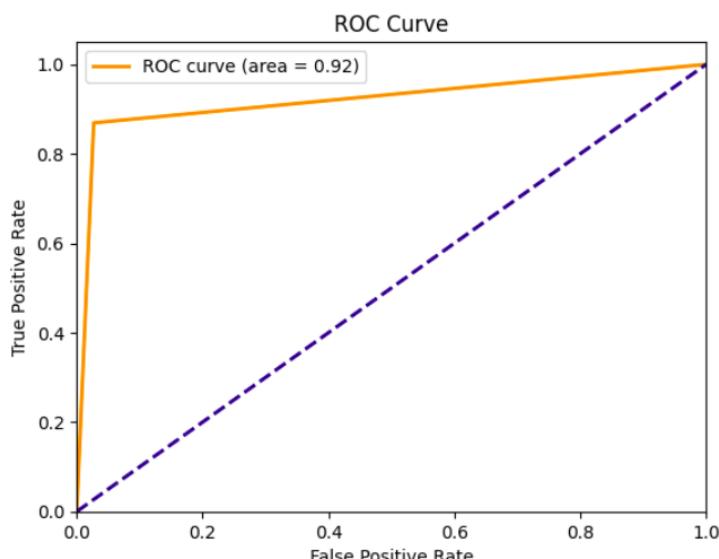
- ۱ - لایه های کانولوشن ابتدا با دقت خوبی ارتباطات بین هر قسمت را کشف میکنند، سپس با کاهش اندازه فیچر ها، آن ها را به مدل LSTM میدهند که وابستگی های زمانی بین داده کشف شود، با حذف لایه های کانولوشنی، عملاً با تعداد زیادی فیچر را به مدل های LSTM میدهیم، و مدل های LSTM در برخورد با ورودی های بزرگ (در اینجا پنجره زمانی  $3^0$ ) با کاهش سرعت و عملکرد رو به رو میشوند.
- ۲ - علاوه بر مورد بالا، تعداد پارامتر ها هم مهم هستند، پارامتر های مدل های ترکیبی قادر به ذخیره و یادگیری بیشتری از دیتا هستند در حالی که مدل هایی که به صورت مجزا آموزش دیده اند، اینطور نیستند.

یک نکته مهم البته، این است که Recall مدل نسبت به CNN افزایش داشته اما با کاهش Precision رو به رو شده است.

مقایسه مدل:

CNN – LSTM Classifier With Early Stopping			LSTM Classifier With Early Stopping	
Class ۰	Class ۱		Class ۰	Class ۱
۰.۹۹	۰.۷۴	Precision	۰.۹۹	۰.۷۱
۰.۹۸	۰.۹۰	Recall	۰.۹۷	۰.۸۷
۰.۹۸	۰.۸۱	F1-Score	۰.۹۸	۰.۷۸
۰.۹۷		Accuracy		۰.۹۷

جدول ۶ - مقایسه مدل LSTM CLs , CNN-LSTM CLs



شکل ۸۷ - منحنی ROC

همچنین در مقایسه AUC این نمودار با نمودار قبلی نیز شاهد کاهش ۲ درصدی سطح زیر نمودار هستیم، از مقدار ۰.۹۴ به ۰.۹۲ رسیده است، بدیهی است که این مدل از مدل ترکیبی ضعیف تر است.

## LSTM – Regressor مدل

معماری مدل که دقیقا طبق مقاله شامل تنها دو لایه Dense است که در ادامه یک لایه LSTM روی آن قرار گرفته است.

```
input_shape = (train_vectors.shape[1], train_vectors.shape[2])
input_layer = Input(shape=input_shape)
x = LSTM(50, return_sequences=True, dropout=0.2)(input_layer)
x = Dropout(0.2)(x)
x = LSTM(50, dropout=0.2)(x)
x = Dropout(0.2)(x)
x = Dense(128, activation='relu')(x)
# classification_output = Dense(1, activation='sigmoid', name='classification_output')(x)
regression_output = Dense(1, activation='linear', name='regression_output')(x)
lstm_regressor_model = Model(inputs=input_layer, outputs=[regression_output]) #outputs=[classification_output, regression_output])
lstm_regressor_model.summary()
```

شکل ۸۸ - مدل LSTM - Regressor

ساختار مدل: که از نظر تعداد پارامتر تقریبا با مدل های قبلی برابر است و قابلیت مقایسه را ممکن میکند.

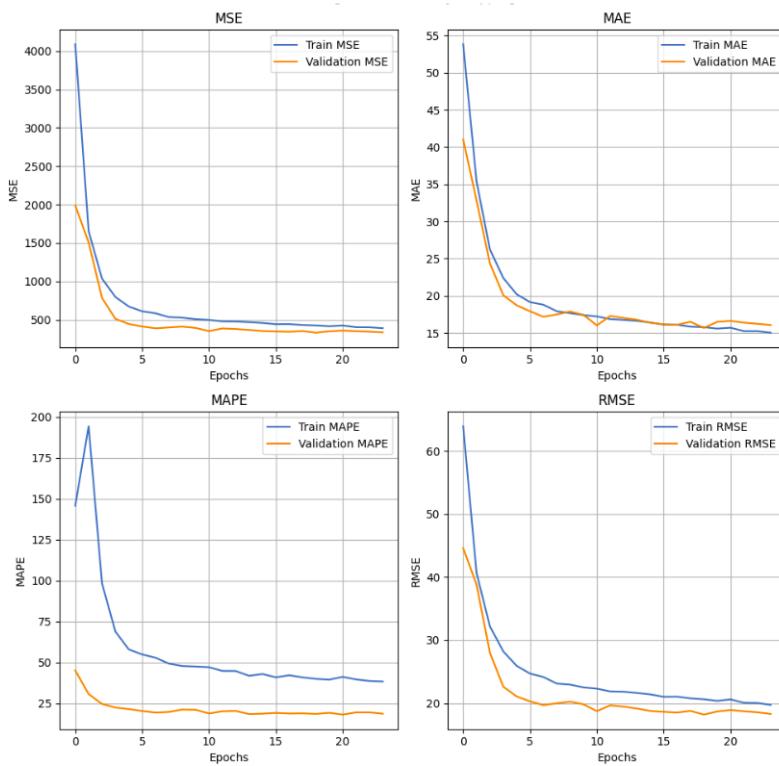
Model: "functional\_11"

Layer (type)	Output Shape	Param #
input_layer_5 (InputLayer)	(None, 30, 18)	0
lstm_6 (LSTM)	(None, 30, 50)	13,800
dropout_6 (Dropout)	(None, 30, 50)	0
lstm_7 (LSTM)	(None, 50)	20,200
dropout_7 (Dropout)	(None, 50)	0
dense_5 (Dense)	(None, 128)	6,528
regression_output (Dense)	(None, 1)	129

```
Total params: 40,657 (158.82 KB)
Trainable params: 40,657 (158.82 KB)
Non-trainable params: 0 (0.00 B)
```

شکل ۸۹ - ساختار مدل LSTM - Regressor

نتایج:



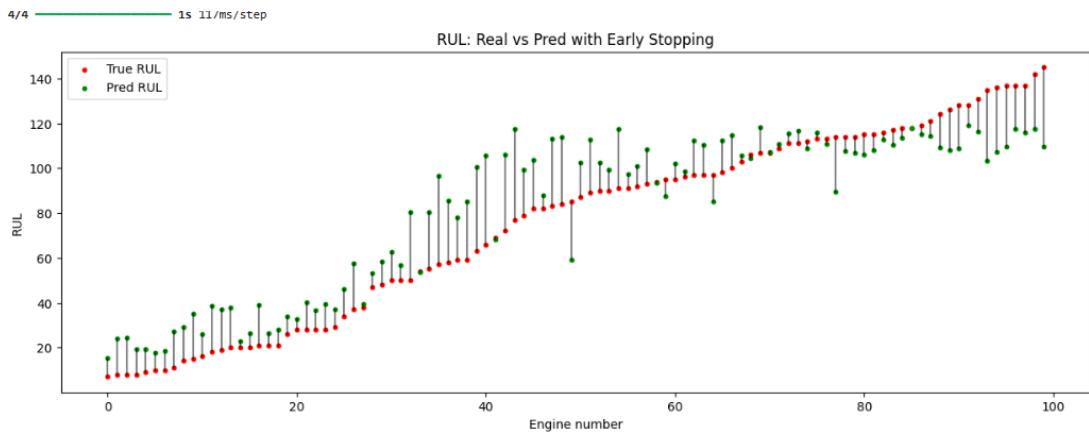
شکل ۹۰ - نمودار های ارزیابی

در نمودارهای بالا، شاهد آموزش خوب مدل هستیم، اگرچه دقت و تابع هزینه به اندازه مدل ترکیبی قبلی مناسب نیستند، اما این موضوع به وضوح قابل مشاهده است.

```
51/51 1s 19ms/step - loss: 301.1899 - mae: 14.9631 - mape: 16.7425 - rmse: 17.3243
all_window MSE: 330.74
all_window MAE: 15.67
all_window MAPE: 18.61
all_window RMSE: 18.19

#####
4/4 1s 21ms/step - loss: 290.7802 - mae: 13.6184 - mape: 32.1257 - rmse: 17.0492
last_window_per_test_data MSE: 297.98
last_window_per_test_data MAE: 13.97
last_window_per_test_data MAPE: 34.39
last_window_per_test_data RMSE: 17.26
```

شکل ۹۱ - نتایج مدل



شکل ۹۲ – مقایسه دیتای واقعی و پیش بینی توسط مدل LSTM - Regressor

مقایسه مدل:

CNN – LSTM Regressor With Early Stopping		LSTM Regressor With Early Stopping		
All-windows	Last-windows		All-windows	Last-windows
۲۹۰.۹۲	۱۸۴.۶۳	MSE	۳۳۰.۷۴	۲۹۷.۹۸
۱۴.۰۹	۱۰.۲۸	MAE	۱۵.۶۷	۱۳.۹۷
۱۴.۲۲	۱۸.۶۸	MAPE	۱۸.۶۱	۳۴.۳۹
۱۷.۰۶	۱۳.۵۹	RMSE	۱۸.۱۹	۱۷.۲۶

جدول ۷ – مقایسه مدل CNN CLs , CNN-LSTM CLs

با مقایسه معیار ها، در حالت Last windows به ازای هر داده تست، که در مقاله نیز همین دیتا مورد بررسی قرار گرفته است، میبینیم مدل پیشنهادی مقاله نسبت به این مدل عملکرد بهتری داشته است، این مورد نیز بدیهی است، همانطور که پیش از این اشاره کردیم، وجود لایه CNN و MAXPooling قبل از لایه LSTM اولاً به ساختن فیچر مپ های در براساس CNN اقدام میکند و این شباهت دیتا را در یک پنجره زمانی بدست می آورد، سپس با لایه های پولینگ این طول دیتا کمتر شده و به مدل LSTM داده میشود، مدل LSTM در اینجا با تقویت فیچر های مرتبط برای پیش بینی آینده و فراموش کردن فیچر های غیرمرتبط، بازنمایی مطلوبی برای لایه های Dense پایانی ایجاد میکند.