



دانشگاه تهران
دانشکده مهندسی برق و
کامپیوتر



درس شبکه‌های عصبی و یادگیری عمیق
تمرین سوم

متین بذرافshan - 810100093

شهریار عطار - 810100186

فهرست

پرسش 1. تحلیل احساسات متن فارسی
1. مجموعه داده 1-1
2. شکل 1-1. توزیع emotion-label‌ها در ستون
2. پیش‌پردازش داده‌ها 1-2
4. نمایش ویژگی 1-3
9. ابعاد پیش‌فرض بردار تعییه در ParsBERT
9. تعداد ابعاد این بردار بیانگر چیست؟
9. Word Embedding مفهوم
10. ساخت مدل 4-1
10. CNN-LSTM .1 مدل
12. شکل 2-1. بهینه‌ساز Adam، نرخ یادگیری $1e-3$ ، سایز دسته 64
12. شکل 2-2. بهینه‌ساز Adam، نرخ یادگیری $1e-4$ ، سایز دسته 64
13. شکل 2-3. بهینه‌ساز SGD، نرخ یادگیری $1e-3$ ، سایز دسته 64
13. شکل 2-4. بهینه‌ساز SGD، نرخ یادگیری $1e-4$ ، سایز دسته 64
14. شکل 2-5. بهینه‌ساز Adam، نرخ یادگیری $1e-3$ ، سایز دسته 8
14. شکل 2-6. بهینه‌ساز Adam، نرخ یادگیری $1e-4$ ، سایز دسته 8
15. شکل 2-7. بهینه‌ساز SGD، نرخ یادگیری $1e-3$ ، سایز دسته 8
15. شکل 2-8. بهینه‌ساز SGD، نرخ یادگیری $1e-4$ ، سایز دسته 8
16. 2. مدل CNN
18. شکل 10-1. معماری مدل CNN
19. شکل 11-1. تغییرات loss و accuracy در هنگام آموزش مدل CNN
20. 3. مدل LSTM
20. شکل 12-1. تغییرات loss و accuracy در هنگام آموزش مدل LSTM
21. مقایسه CNN و LSTM
21. شبکه CNN
21. شبکه LSTM
22. ادغام CNN و LSTM
23. 1. مدل CNN-LSTM 5-1
24. 2. مدل CNN
24. 3. مدل LSTM
27. 6-1. امتیازی

28.....	Gaussian Naive Bayes: .1
29.....	Decision Tree: .2
30.....	Gradient Boosting: .3
30.....	Random Forest: .4
31.....	Logistic Regression: .5
33.....	پرسش 2 - سامانه‌های سایبرفیزیکی: نگهداری هوشمند
33.....	1-2 پیش‌پردازش داده‌ها
35.....	شكل 2-1. توزیع تعداد cycle-ها در دیتاست train
36.....	Data Selection .1
37.....	شكل 2-2. اعداد ثبت شده توسط سنسورها با نویز در طول زمان برای 10 موتور
38.....	شكل 2-3. اعداد ثبت شده توسط سنسورها بدون نویز (میانگین در یک window) در طول زمان برای 10 موتور
39.....	Data Labeling .2
40.....	Data Normalization .3
40.....	Time Windowing .4
41.....	2-2. مدل‌سازی و ارزیابی
41.....	4-2. معماری مدل پیشنهادی
42.....	شكل 2-5. معماری مدل پیشنهادی با جزئیات بیشتر
43.....	Classification
45.....	شكل 2-6. تفاوت لایه‌های Conv2D و Conv1D
45.....	شكل 2-6. نمودار تغییر loss در زمان آموزش
46.....	بدون early stopping
47.....	شكل 2-7-2. Confusion Matrix برای مدل هایبرید classification پیشنهادی بدون early stopping
47.....	شكل 2-8. نمودار ROC برای مدل هایبرید classification پیشنهادی بدون early stopping
48.....	شكل 2-9. نمودار آموزش برای مدل هایبرید classification پیشنهادی بدون early stopping
49.....	با early stopping
50.....	شكل 2-10-2. Confusion Matrix برای مدل هایبرید classification پیشنهادی با early stopping
50.....	شكل 2-11. نمودار ROC برای مدل هایبرید classification پیشنهادی بدون early stopping
51.....	شكل 2-12. نمودار آموزش برای مدل هایبرید classification پیشنهادی بدون early stopping

52.....	Regression
54.....	بدون early stopping
54.....	با early stopping
55.....	نتایج
55.....	شکل 2-14. نمودار آموزش برای مدل هایبرید regression پیشنهادی
56.....	در حالت استفاده از تمام window-ها
.....	شکل 2-15. نمودار RUL-های واقعی و تخمین زده شده در حالت استفاده از تمام
56.....	window-ها برای مدل هایبریدی پیشنهادی
57.....	در حالت استفاده تنها از آخرین window-ها
.....	شکل 2-16. نمودار RUL-های واقعی و تخمین زده شده در حالت استفاده از آخرین
57.....	window برای مدل هایبریدی پیشنهادی
58.....	3. مقایسه با مدل‌های پایه
58.....	LSTM
59.....	Classification
60.....	شکل 17-2. LSTM classification برای مدل Confusion Matrix
61....	شکل 2-19. نمودار ROC برای مدل LSTM classification بدون
61.....	شکل 2-20. نمودار آموزش برای مدل LSTM regression
62.....	Regression
63.....	شکل 2-21. نمودار آموزش برای مدل LSTM regression
64.....	در حالت استفاده از تمام window-ها
.....	شکل 2-22. نمودار RUL-های واقعی و تخمین زده شده در حالت استفاده از تمام
64.....	window-ها برای مدل LSTM
65.....	در حالت استفاده تنها از آخرین window-ها
.....	شکل 2-23. نمودار RUL-های واقعی و تخمین زده شده در حالت استفاده از آخرین
65.....	window برای مدل LSTM
66.....	CNN
67.....	Classification
68.....	شکل 24-2. CNN classification برای مدل Confusion Matrix
68.....	شکل 2-25. نمودار ROC برای مدل CNN classification
68.....	شکل 2-26. نمودار آموزش برای مدل CNN regression
69.....	Regression
70.....	شکل 2-27. نمودار آموزش برای مدل CNN regression
71.....	در حالت استفاده از تمام window-ها
.....	شکل 2-28. نمودار RUL-های واقعی و تخمین زده شده در حالت استفاده از تمام
71.....	window-ها برای مدل CNN

72.....	در حالت استفاده تنها از آخرين window-ها
شکل 2-29. نمودار RUL-های واقعی و تخمین زده شده در حالت استفاده از آخرين	
72.....	CNN برای مدل window
73.....	مقایسه
73.....	شکل 2-30. مقایسه مدل‌های regression با تمام window-ها
73.....	شکل 2-31. مقایسه مدل‌های regression با آخرین window
73.....	شکل 2-32. مقایسه مدل‌های classification با تمام window-ها
73.....	شکل 2-33. مقایسه مدل‌های classification با آخرین window

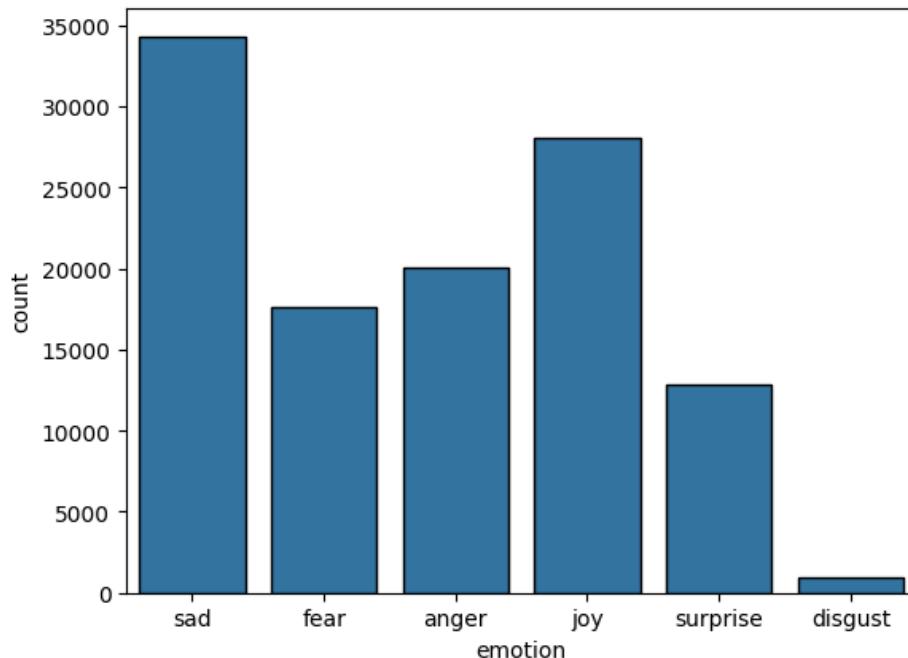
پرسش 1. تحلیل احساسات متن فارسی

1-1. مجموعه داده

دیتاست داده شده حاصل از `scrape` کردن توییتر به کمک `snsccape` است که بر اساس `Ekman's 6 main emotions` به صورت دستی `label` زده شده است. به منظور حفظ حریم شخصی تمام `username`-ها و بقیه اطلاعات نیز از دیتاست حذف شده و دیتاست شامل اطلاعات زیر می‌باشد:

tweet (1
replyCount (2
retweetCount (3
likeCount (4
quoteCount (5
hashtags (6
sourceLabel (7
emotion (8

که ما در این سوال فقط با ستون‌های `tweet` و `emotion` کار داریم. توزیع `label`-ها در ستون `emotion` می‌توانید در شکل زیر مشاهده کنید.



شکل 1-1. توزیع emotion-label-ها در ستون

2-2. پیشپردازش داده‌ها

برای پیشپردازش داده‌های متنی داده شده، کارهای زیر انجام شده است:

و شامل @ut و شامل یوزرنیم www.ut.ac.ir سلام این متن شامل تگ </> و شامل لینک #هشتگ و شامل ایموجی 😊 و شامل علائم نگارشی! و شامل کلمات توقف مثل به و و شامل تکراراً حروف و شامل نیمفاصله است.

1. حذف تگ‌های HTML

و شامل یوزرنیم www.ut.ac.ir سلام این متن شامل تگ و شامل لینک remove url: و شامل #هشتگ و شامل ایموجی 😊 و شامل علائم نگارشی! و شامل کلمات توقف @ut مثل به و و شامل تکراراً حروف و شامل نیمفاصله است.

2. حذف لینک‌ها

و شامل یوزرنیم www.ut.ac.ir سلام این متن شامل تگ و شامل لینک remove url: و شامل #هشتگ و شامل ایموجی 😊 و شامل علائم نگارشی! و و شامل کلمات توقف @ut مثل به و و شامل تکراراً حروف و شامل نیمفاصله است.

3. حذف یوزرنیم‌ها

سلام این متن شامل تگ و شامل لینک و شامل یوزرنیم و شامل `remove username:` هشتگ و شامل ایموجی 😊 و شامل علائم نگارشی! و و شامل کلمات توقف مثل به و و شامل تکراراً حروف و شامل نیمه‌فاصله است.

4. حذف علامت هشتگ، خود کلمه نگه داشته می‌شود.

سلام این متن شامل تگ و شامل لینک و شامل یوزرنیم و شامل `remove hashtag:` هشتگ و شامل ایموجی 😊 و شامل علائم نگارشی! و و شامل کلمات توقف مثل به و و شامل تکراراً حروف و شامل نیمه‌فاصله است.

5. اصلاح تکرار حروف

سلام این متن شامل تگ و شامل لینک و شامل یوزرنیم و شامل `remove rep:` هشتگ و شامل ایموجی 😊 و شامل علائم نگارشی! و و شامل کلمات توقف مثل به و و شامل تکرار حروف و شامل نیمه‌فاصله است.

6. حذف علائم نگارشی

سلام این متن شامل تگ و شامل لینک و شامل یوزرنیم و شامل `remove punctuation:` هشتگ و شامل ایموجی 😊 و شامل علائم نگارشی و و شامل کلمات توقف مثل به و و شامل تکرار حروف و شامل نیمه‌فاصله است

7. جایگزینی ایموجی استفاده شده با کلمه هم‌معنی آن

سلام این متن شامل تگ و شامل لینک و شامل یوزرنیم و شامل `demojize:` ایموجی : خنده: و شامل علائم نگارشی و و شامل کلمات توقف مثل به و و شامل تکرار حروف و شامل نیمه‌فاصله است

8. حذف کلمات توقف^۱

سلام متن تگ لینک یوزرنیم هشتگ ایموجی : خنده : علائم نگارشی : کلمات توقف تکرار حروف نیم فاصله

^۱ Stop Words

9. بازگردانی کلمات به ریشه آن‌ها²

سلا متن تگ لینک یوزرن هشتگ ایموج خنده علائم نگار کل توقف تکرار حروف stemming: نیم فاصله

10. جایگزینی نیم‌فاصله با فاصله

3-1. نمایش ویژگی

توكنایزر³ در مدل‌های NLP یک ابزار حیاتی است که متن ورودی را به واحدهای کوچکتری به نام توکن‌ها تقسیم می‌کند. ParsBERT⁴ یک مدل برای زبان فارسی است. Tokenizer نقش مهمی در آماده‌سازی متن برای ورود به مدل دارد و کاربردهای مختلفی دارد، از جمله:

1. **پیش‌پردازش متن:** توكنایزر، متن را به توکن‌های قابل درک برای مدل تبدیل می‌کند. این توکن‌ها می‌توانند کلمات، تکوازها یا حتی کاراکترها باشند.

2. **مدیریت وکتورهای ورودی:** هر توکن به یک وکتور عددی تبدیل می‌شود که مدل می‌تواند آن را پردازش کند. این کار به مدل کمک می‌کند تا روابط معنایی بین توکن‌ها را درک کند.

3. **حذف نویزها و علائم زائد:** Tokenizer می‌تواند علائم نگارشی، اعداد و نویزهای غیرضروری را حذف کند یا آن‌ها را به صورت خاصی پردازش کند که مدل بتواند به درستی روی متن اصلی تمرکز کند.

4. **مدیریت توکن‌های خاص:** توکن‌های خاص مانند [CLS] (برای شروع جمله) و [SEP] (برای جداسازی جملات) را اضافه می‌کند که برای مدل‌های ترانسفورمر مانند BERT ضروری هستند.

5. **پشتیبانی از تکوازها:** در زبان‌هایی مانند فارسی که ممکن است کلمات پیچیده و ترکیبی زیادی داشته باشند، Tokenizer می‌تواند کلمات را به تکوازها تقسیم کند که این کار دقیق را افزایش می‌دهد.

² Stemming

³ Tokenizer

⁴ Bidirectional Encoder Representations from Transformers

توكنایزر ParsBERT به طور خاص برای زبان فارسی بهینه‌سازی شده است و از این رو می‌تواند در کارهای مختلفی مانند ترجمه ماشینی، تحلیل احساسات، پاسخ به سوالات، خلاصه‌سازی متن و غیره بسیار موثر باشد.

برای نشان دادن چگونگی تبدیل متن به عدد توسط مدل ParsBERT، ابتدا باید مراحل پردازش متن را توسط توكنایزر و سپس تبدیل آن به وکتورهای تعبیه⁵ شرح دهیم.

در این مثال، از جمله فارسی زیر استفاده می‌کنیم:

"من به کتابخانه می‌روم."

1. توكنایز

ابتدا، جمله به توکن‌ها تقسیم می‌شود. فرض می‌کنیم که از یک Tokenizer ساده استفاده می‌کنیم که هر کلمه را به عنوان یک توکن جدا می‌کند:

["من" , "به" , "کتابخانه" , "می‌روم" , ":"] -

با این حال، توكنایزر ParsBERT از توکن‌های زیرکلمه‌ای استفاده می‌کند که ممکن است کلمات را به بخش‌های کوچکتر تقسیم کند. فرض کنید توکن‌ها به صورت زیر تبدیل شوند:

"من" [من] -

[به] → "به" -

"کتابخانه" → [کتاب، خانه] -

"می‌روم" → [می، روم] -

[.] → ":" -

2. نگاشت به شناسه‌های عددی⁶

سپس، هر توکن به یک شناسه عددی تبدیل می‌شود. این شناسه‌ها از یک لغتنامه⁷ از پیش‌آموزش دیده استخراج می‌شوند. فرض می‌کنیم فرهنگ لغت ما به این صورت باشد:

"من" → 1023 -

⁵ Embeddings

⁶ Numerical ID

⁷ Vocabulary

- 1056 → "به"

- 2001 → "کتاب"

- 2050 → "خانه"

- 1800 → "می"

- 1900 → "روم"

- 500 → ":"

بنابراین، جمله "من به کتابخانه می‌روم." به شناسه‌های عددی زیر تبدیل می‌شود:

[500, 1900, 1800, 2050, 2001, 1056, 1023] -

3. توکن‌های ویژه

مدل‌های BERT نیاز به توکن‌های خاصی مانند [CLS] برای شروع و [SEP] برای پایان جمله دارند. پس از اضافه کردن این توکن‌ها، دنباله نهایی به این صورت خواهد بود:

- [CLS] + [SEP] + [.، می، #روم، #خانه، #کتاب، به، من]

با شناسه‌های عددی، دنباله به صورت زیر است:

- [101, 102, 1023, 1056, 1021, 1056, 1023, 101] -

4. تبدیل به بردار تعبیه

در مرحله بعد، هر یک از این شناسه‌های عددی به یک وکتور عددی تبدیل می‌شوند. این وکتورها توسط مدل از پیشآموزش دیده و در طول آموزش بهبود می‌یابند.

5. نتیجه نهایی:

- متن ورودی: "من به کتابخانه می‌روم."

- توکن‌ها: [CLS] من به کتاب #خانه می #روم . [SEP]

- شناسه‌های عددی: [101, 102, 1023, 1056, 1021, 1056, 1023, 101] -

- وکتورهای عددی: یک ماتریس با ابعاد (تعداد توکن‌ها، ابعاد وکتورها)

حال از آنجا که ممکن است طول داده‌ها با هم متفاوت باشد ما حداکثر طول را 32 در نظر گرفتیم و در صورتی که طول جمله از 32 کمتر بود از حاشیه‌گذاری⁸ استفاده کردیم.

حاشیه‌گذاری:

حاشیه‌گذاری یا پدینگ، یک روش برای تصحیح اندازه یک بردار می‌باشد. معمولاً ابعاد بردارها در در طول شبکه دچار تغییراتی می‌شوند، از طرفی اندازه ورودی هر لایه ثابت می‌باشد و در طول استفاده از شبکه نمی‌توان آن را تغییر داد. به همین علت حاشیه‌گذاری یک روش کارآمد برای رسیدگی این موضوع می‌باشد. حاشیه‌گذاری انواع مختلفی دارد، ما از حاشیه‌گذاری صفر⁹ استفاده کردیم. در این روش به اطراف بردار مقادیر صفر اضافه می‌شود تا ابعاد یک بردار به اندازه مورد نیاز برسد.

در طول ساخت بردار دو بعدی مربوط به جملات، اگر جمله کوتاه بود و توکن‌های آن پیش از رسیدن به انتهای ماتریس (که سایز آن را از پیش تعیین کرده‌ایم) تمام شد، در ادامه بردارهای صفر اضافه کردیم تا طول ماتریس به طول مشخص شده برسد. همچنین اگر تعداد توکن‌های یک جمله بیشتر از مقدار مشخص شده بود، ادامه‌ی توکن‌ها را حذف کردیم.

دقت کنید، دوبار از مدل ParsBert استفاده می‌شود. ابتدا برای توکنایز کردن و سپس برای ساختن وکتورهای تعبیه.

1. **توکنایز کردن:** توکنایز ParsBert به ازای هر کلمه سه مقدار برمی‌گرداند که هر کدام را به ترتیب در لیست مربوطه ذخیره کردیم:
 - a. **شناسه:** شناسه نظیر هر عدد در لغتنامه
 - b. **ماسک:** نشان‌دهنده این است که آیا این توکن در جمله بوده‌است یا خیر، یک به معنی کلمه و صفر به معنی پدینگ است.
 - c. **تایپ:** نشان‌دهنده تعلق یک توکن به جمله است.

2. **بردارهای تعبیه:** برای این کار هر سه لیست ساخته‌شده توسط توکنایزر برای هر جمله را به مدل تنسور ParsBert¹⁰ می‌دهیم. دقت کنید که قبل از بارگیری مدل، باید ابعاد خروجی را به 120 تغییر دهیم. این مدل خود یک شبکه عصبی با 12 لایه است که با

⁸ Padding

⁹ Zero Padding

¹⁰ TFAutoModel

توجه به تمامی توکن‌های موجود در یک جمله، بردارهای عددی را می‌سازد. به علت زیاد بودن تعداد نمونه‌ها و همچنین محدود بودن حافظه، داده‌ها را به دسته‌های مختلف تقسیم کردیم و خروجی شبکه Bert را برای هر دسته جداگانه حساب می‌کردیم و سپس Garbage Collector جواب‌ها را به یکدیگر الحق^{۱۱} کردیم. بعد از اتمام اجرا هر دسته از برای پاکسازی حافظه موقت استفاده می‌کردیم.

تفاوت عمدۀ این مدل با مدل Word2Vec در همین مورد است. در مدل Word2Vec برای هر کلمه، همواره یک بردار عددی نظیر آن وجود دارد، اما در مدل Bert برای هر کلمه با توجه به کلمه‌های اطراف آن و بافت جمله^{۱۲} تصمیم‌گیری می‌شود. برای مثال دو جمله زیر را درنظر بگیرید.

- a. من به کتابخانه کنار بانک رفتم.
- b. دزد از بانک ذخیره‌های ارزی را دزدید.

در این دو جمله "بانک" معنی یکسانی از لحاظ مفهومی ندارد. در جمله اول از بانک تنها برای مشخص کردن مکان استفاده شده است، اما در جمله دوم خود بانک نیز مدنظر بوده است.

¹¹ Concatenate

¹² Context

ابعاد پیشفرض بردار تعبیه در ParsBERT

ابعاد پیشفرض Embedding در مدل ParsBERT معمولاً 768 است. این عدد نشان‌دهنده تعداد ویژگی‌هایی است که هر توکن به وکتور عددی تبدیل می‌شود. در واقع، هر کلمه یا توکن در مدل ParsBERT به یک وکتور 768 بعدی تبدیل می‌شود.

تعداد ابعاد این بردار بیانگر چیست؟

تعداد ابعاد بردار تعبیه نشان‌دهنده تعداد ویژگی‌هایی است که مدل برای نمایش هر توکن استفاده می‌کند. هر یک از این 768 عدد می‌تواند یک ویژگی خاص از توکن را نشان دهد، مانند معنای لغوی، نقش دستوری، موقعیت در جمله و ارتباط معنایی با توکن‌های دیگر. این ابعاد به مدل کمک می‌کنند تا بتواند پیچیدگی و تنوع معنایی زبان را بهتر درک کند.

Word Embedding مفهوم

Word Embedding یک نمایش عددی از کلمات است که ویژگی‌های معنایی و نحوی کلمات را در فضای چندبعدی حفظ می‌کند. بردارهای تعبیه به گونه‌ای آموزش داده می‌شوند که کلمات با معنای مشابه یا کاربرد مشابه در فضای برداری به یکدیگر نزدیک باشند. این کار با استفاده از تکنیک‌های یادگیری عمیق و شبکه‌های عصبی انجام می‌شود. بردارهای تعبیه به مدل‌های پردازش زبان طبیعی کمک می‌کنند تا:

- **کلمات مشابه را تشخیص دهند:** کلمات با معنای مشابه بردارهای تعبیه نزدیک به هم خواهند داشت.
- **ارتباطات معنایی را درک کنند:** مدل می‌تواند روابط معنایی پیچیده بین کلمات را شناسایی کند.
- **وظایف مختلف NLP را انجام دهند:** مانند طبقه‌بندی متن، تحلیل احساسات، ترجمه ماشینی و غیره.

4-1. ساخت مدل

بعد از آماده‌سازی بردارهای تعبیه، حال داده‌ها را به سه بخش آموزش¹³، اعتبارسنجی¹⁴ و آزمون¹⁵ تقسیم کرده‌ایم. همانگونه که ذکر شده بود 56 درصد به آموزش، 14 درصد به اعتبارسنجی و 30 درصد به آزمون تخصیص داده‌ایم.

در ادامه، سه نوع مدل ساخته‌ایم.

1. مدل CNN-LSTM

ابتدا سه لایه کانولوشنی یک‌بعدی قرار دادیم، در هر لایه جلوتر، تعداد فیلترها و اندازه کرنل را افزایش دادیم. سپس از یک لایه LSTM استفاده کردیم و در انتها لایه‌های کاملاً متصل¹⁶ برای طبقه‌بندی¹⁷ استفاده کردیم.

```
def cnn_lstm():
    model = Sequential()
    model.add(Layers.Input(shape=(32, 120, )))
    model.add(Layers.Conv1D(filters=64, kernel_size=3,
activation='relu', padding='same'))
    model.add(Layers.Conv1D(filters=128, kernel_size=5,
activation='relu', padding='same'))
    model.add(Layers.Conv1D(filters=256, kernel_size=7,
activation='relu', padding='same'))
    model.add(Layers.MaxPooling1D(pool_size=2, padding='same'))
    model.add(Layers.LSTM(100, return_sequences=True))
    model.add(Layers.Flatten())
    model.add(Layers.Dense(256, activation='relu'))
    model.add(Layers.BatchNormalization())
    model.add(Layers.Dropout(0.3))
    model.add(Layers.Dense(128, activation='relu'))
    model.add(Layers.BatchNormalization())
    model.add(Layers.Dropout(0.3))
    model.add(Layers.Dense(6, activation='softmax'))

    return model
```

¹³ Train

¹⁴ Validation

¹⁵ Test

¹⁶ Fully Connected

¹⁷ Classification

جستوجوی حریصانه:

برای پیدا کردن پارامترهای بهینه آموزش، از جستوجوی حریصانه استفاده کردیم. این کار را برای سه پارامتر نرخ آموزش، بهینه‌ساز و اندازه بچ انجام داده‌ایم. مقادیر این پارامترها در جدول زیر آمده است:

optimizer	learning rate	batch size
Adam ¹⁸	1e-3	8
SGD ¹⁹	1e-4	64

در ادامه از Categorical Cross-Entropy برای تابع هزینه²⁰ استفاده کردیم. همچنین تعداد دوره²¹‌ها را متغیر قرار دادیم اما از Early Stopping استفاده کردیم، به طوری که اگر بعد از 3 دوره مقدار دقیق داده‌های صحت‌سنجی افزایش پیدا نکند، یادگیری متوقف می‌شود.

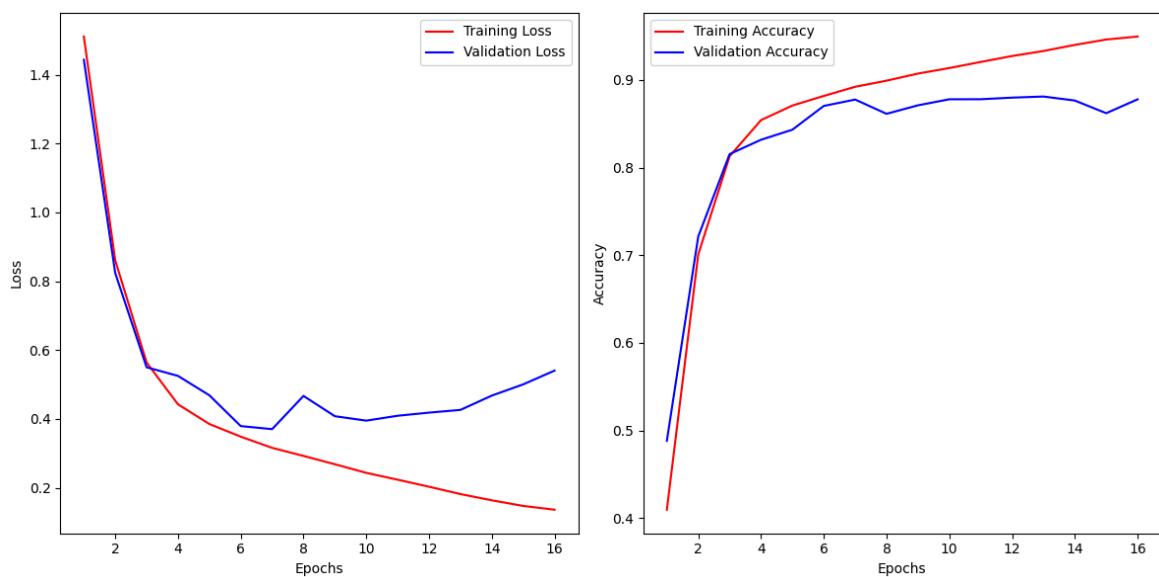
در ادامه نمودارهای آموزش‌های مختلف را می‌توانید مشاهده کنید:

¹⁸ Adaptive Moment Estimation

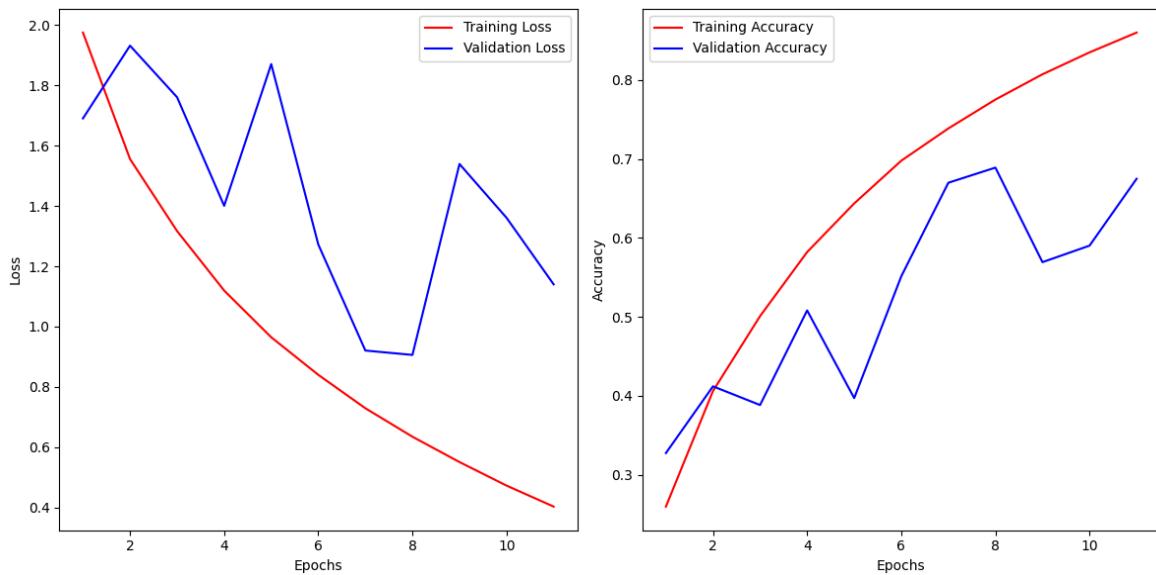
¹⁹ Stochastic Gradient Descent

²⁰ Loss Function

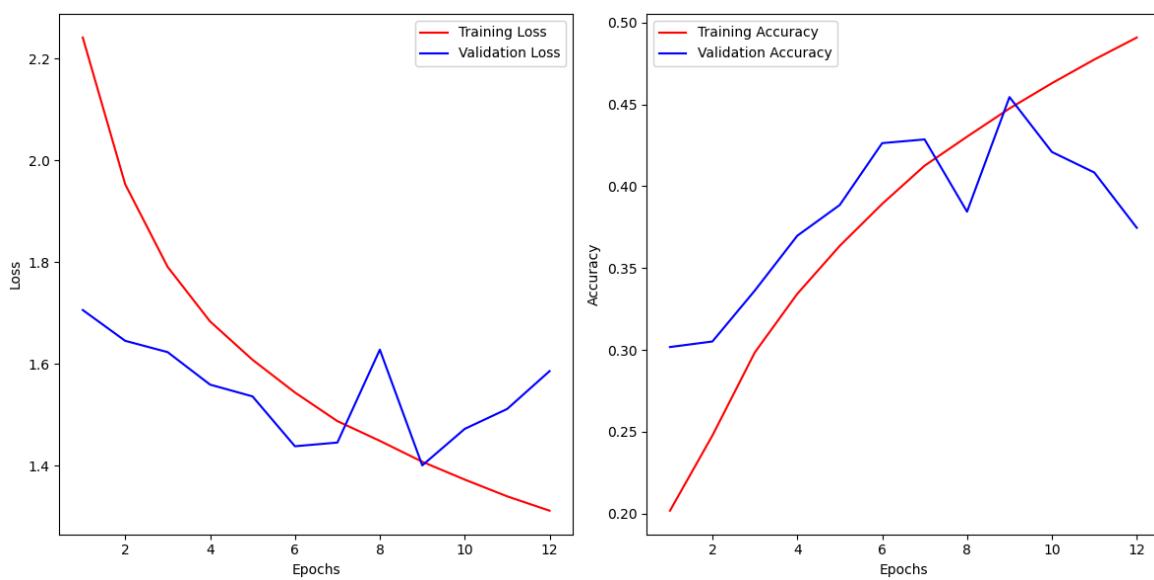
²¹ Epoch



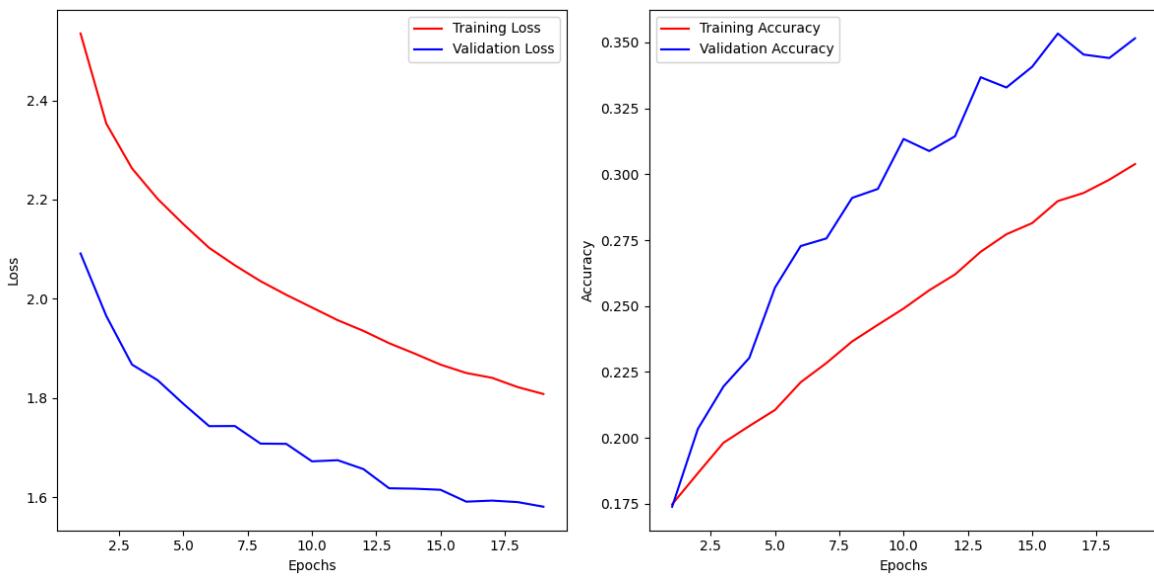
شکل 2-1. بهینهساز Adam، نرخ یادگیری $1e-3$ ، سایز دسته 64



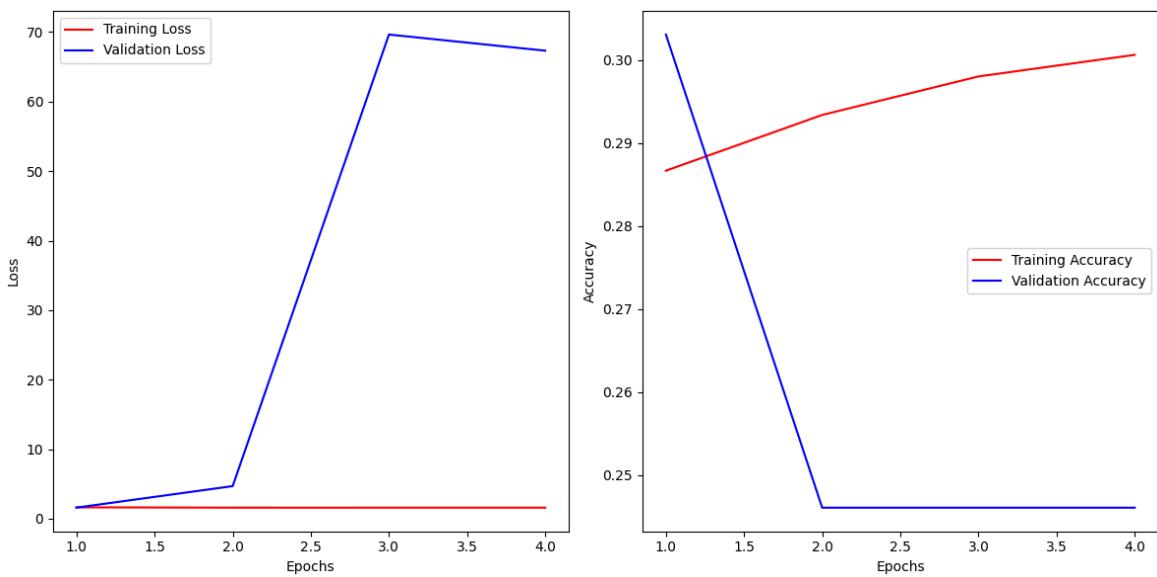
شکل 2-3. بهینهساز Adam، نرخ یادگیری $1e-4$ ، سایز دسته 64



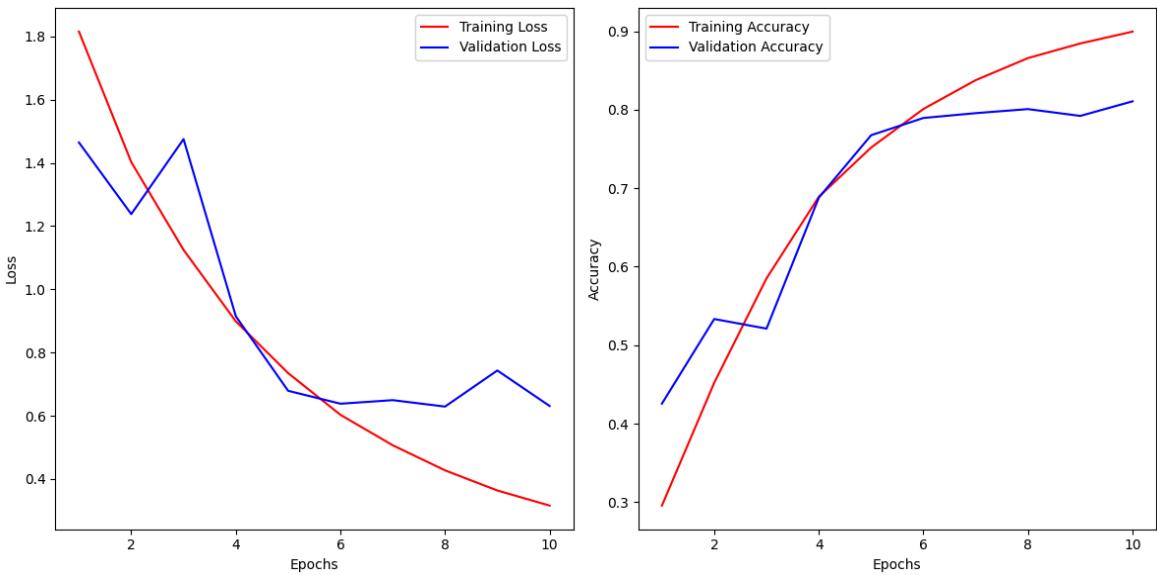
شکل 4-1. بهینهساز SGD، نرخ یادگیری $1e-3$ ، سایز دسته 64



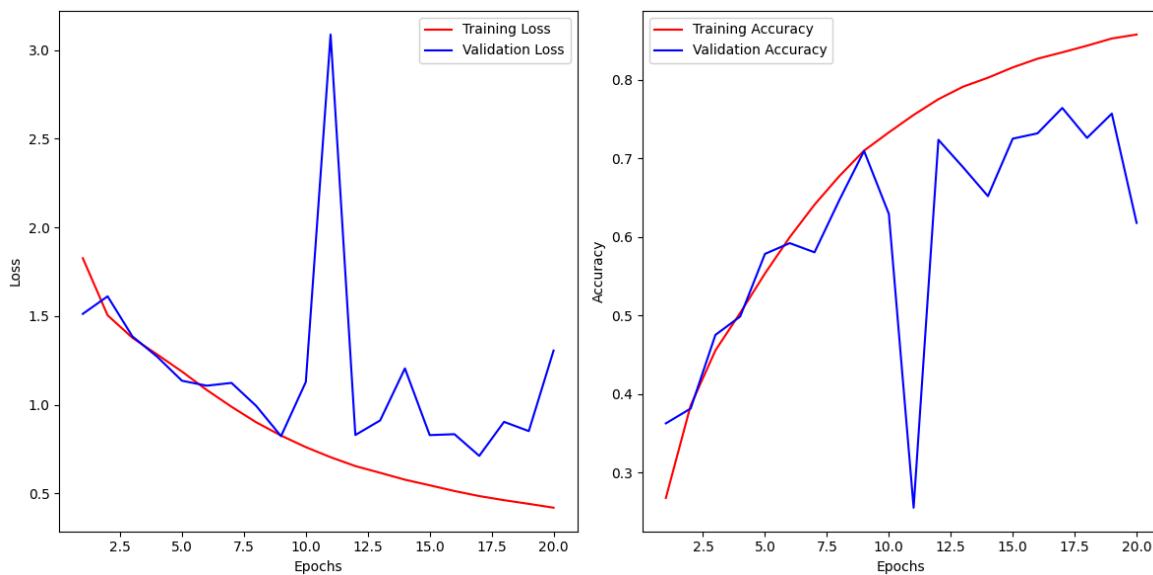
شکل 5-1. بهینهساز SGD، نرخ یادگیری $1e-4$ ، سایز دسته 64



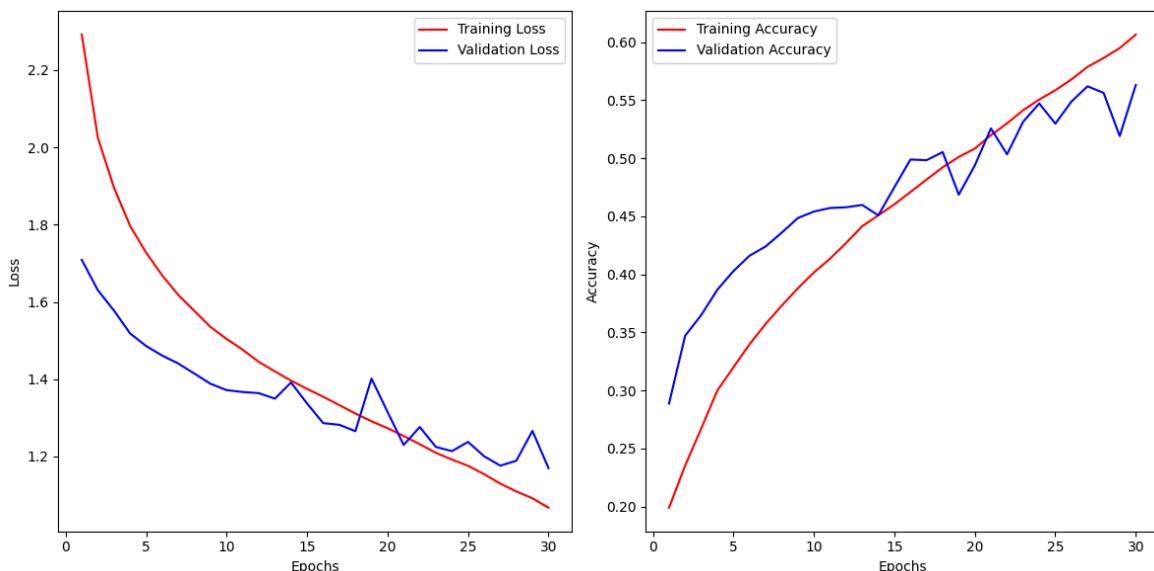
شکل 6-1. بهینه‌ساز Adam، نرخ یادگیری $1e-3$ ، سایز دسته 8



شکل 7-1. بهینه‌ساز Adam، نرخ یادگیری $1e-4$ ، سایز دسته 8



شکل 1-8. بهینه‌ساز SGD، نرخ یادگیری $1e-3$ ، سایز دسته 8



شکل 1-9. بهینه‌ساز SGD، نرخ یادگیری $1e-4$ ، سایز دسته 8

پارامترهای بهینه:

همانگونه مشاهده کردید، پارامترهای بهینه‌ساز Adam با نرخ یادگیری $1e-3$ و سایز دسته 64، بیشترین دقیقت را کسب کردند. جزئیات نتیجه پیش‌بینی شبکه روی داده‌ی آزمون را می‌توانید در بخش بعد ببینید.

CNN .2 مدل

در این بخش، کمی مدل CNN را پیچیده‌تر از قبل کردیم. سه دسته لایه را به طور موازی با کرنل سایز مختلف قرار دادیم. هر دسته شامل سه لایه کانولوشنی است که هر لایه از لایه قبلی تعداد بیشتری فیلتر دارد و همچنین سایز کرنل آن بزرگ‌تر است.

```
def cnn():
    inputs = Layers.Input(shape=(32, 120,))

    conv1 = Layers.Conv1D(filters=32, kernel_size=3,
activation='relu', padding='same')(inputs)
    conv1 = Layers.BatchNormalization()(conv1)
    conv1 = Layers.Conv1D(filters=32, kernel_size=3,
activation='relu', padding='same')(conv1)
    conv1 = Layers.BatchNormalization()(conv1)
    conv1 = Layers.MaxPooling1D(pool_size=2,
padding='same')(conv1)
    conv1 = Layers.Dropout(0.2)(conv1)

    conv2 = Layers.Conv1D(filters=64, kernel_size=5,
activation='relu', padding='same')(inputs)
    conv2 = Layers.BatchNormalization()(conv2)
    conv2 = Layers.Conv1D(filters=64, kernel_size=5,
activation='relu', padding='same')(conv2)
    conv2 = Layers.BatchNormalization()(conv2)
    conv2 = Layers.MaxPooling1D(pool_size=2,
padding='same')(conv2)
    conv2 = Layers.Dropout(0.2)(conv2)

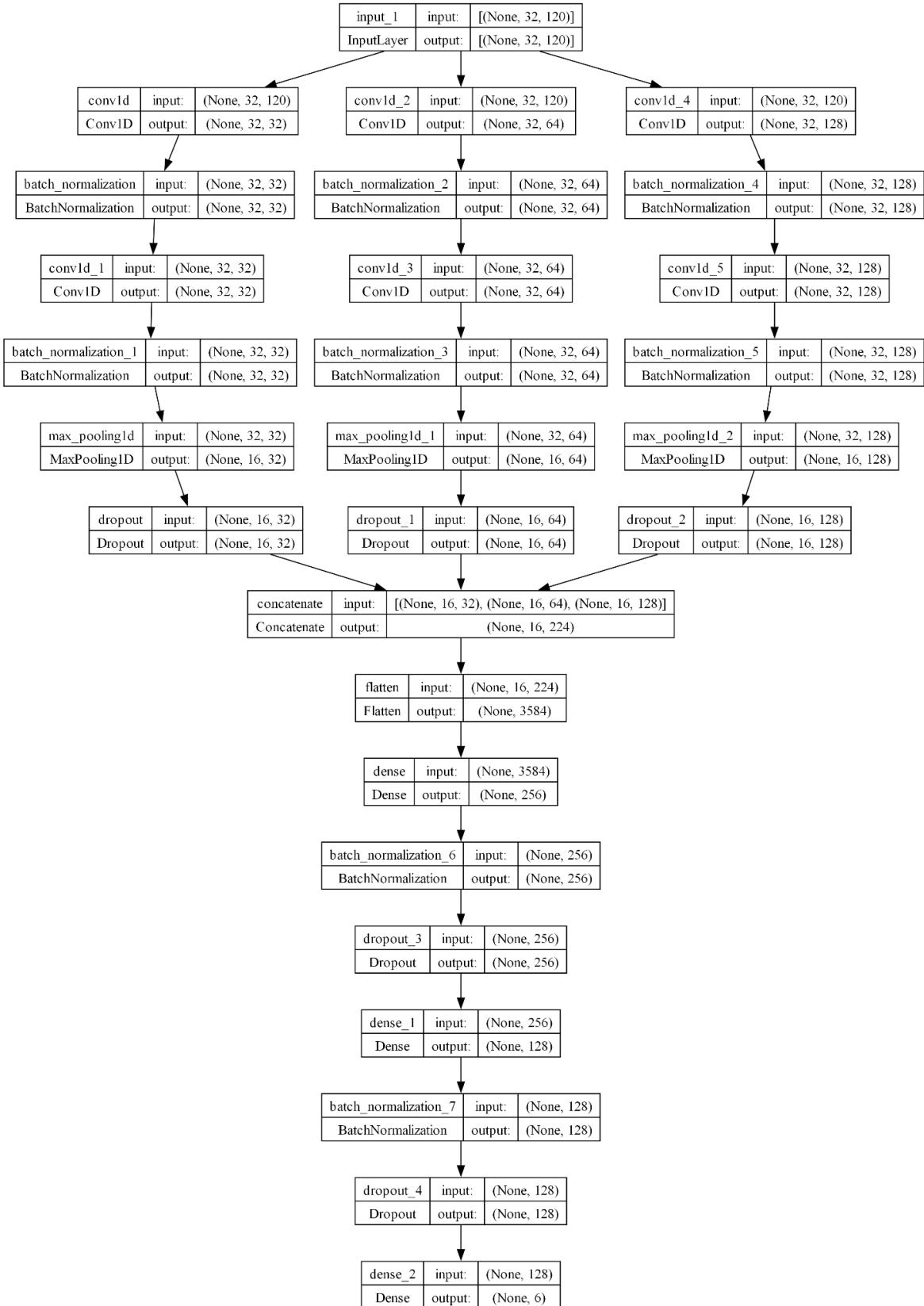
    conv3 = Layers.Conv1D(filters=128, kernel_size=7,
activation='relu', padding='same')(inputs)
    conv3 = Layers.BatchNormalization()(conv3)
    conv3 = Layers.Conv1D(filters=128, kernel_size=7,
activation='relu', padding='same')(conv3)
```

```
conv3 = Layers.BatchNormalization()(conv3)
conv3 = Layers.MaxPooling1D(pool_size=2,
padding='same')(conv3)
conv3 = Layers.Dropout(0.2)(conv3)

merged = Layers.concatenate([conv1, conv2, conv3])
flattened = Layers.Flatten()(merged)

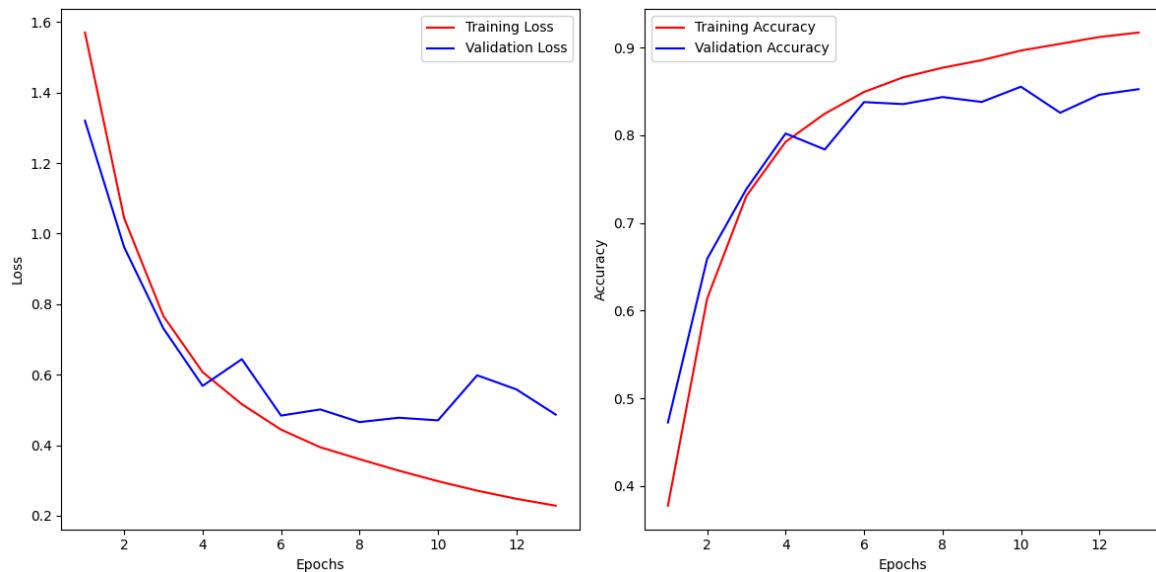
dense1 = Layers.Dense(256, activation='relu')(flattened)
dense1 = Layers.BatchNormalization()(dense1)
dense1 = Layers.Dropout(0.3)(dense1)
dense2 = Layers.Dense(128, activation='relu')(dense1)
dense2 = Layers.BatchNormalization()(dense2)
dense2 = Layers.Dropout(0.3)(dense2)
outputs = Layers.Dense(6, activation='softmax')(dense2)

model = Model(inputs=inputs, outputs=outputs)
return model
```



شکل 10-1. معماری مدل CNN

این مدل را با پارامترهای بهینه به دست آمده از بخش قبلی اجرا کردیم، می‌توانید نتایج حاصل شده را در ادامه مشاهده کنید.



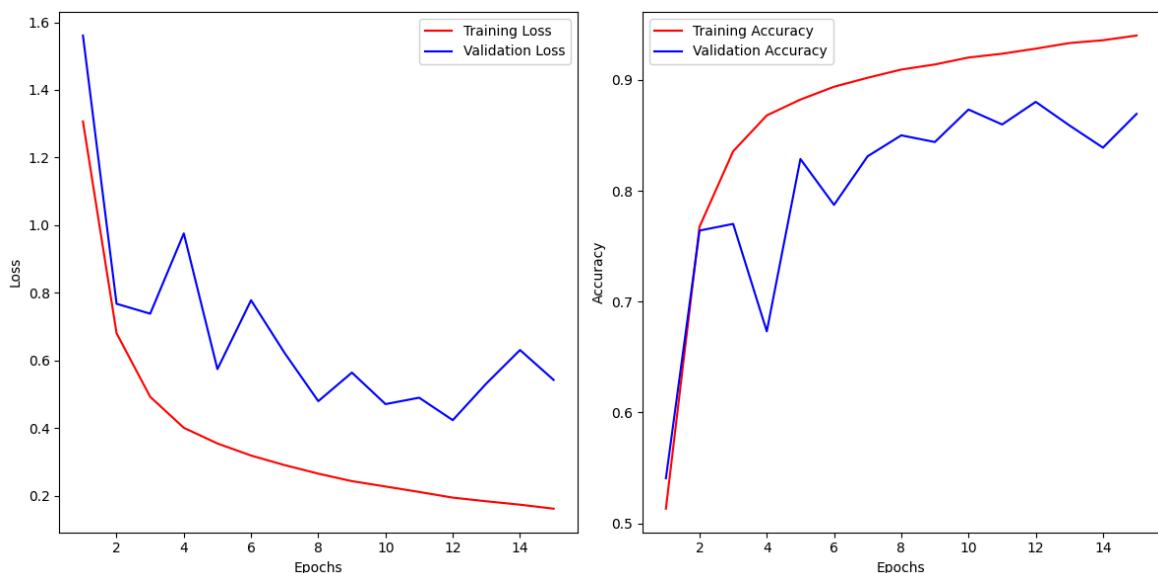
شکل 11-1. تغییرات accuracy و loss در هنگام آموزش مدل CNN

3. مدل LSTM

در این بخش، تنها از یک لایه ساده LSTM استفاده کردیم.

```
def lstm():
    model = Sequential()
    model.add(Layers.Input(shape=(32, 120, )))
    model.add(Layers.LSTM(100, return_sequences=True))
    model.add(Layers.Flatten())
    model.add(Layers.Dense(256, activation='relu'))
    model.add(Layers.BatchNormalization())
    model.add(Layers.Dropout(0.3))
    model.add(Layers.Dense(128, activation='relu'))
    model.add(Layers.BatchNormalization())
    model.add(Layers.Dropout(0.3))
    model.add(Layers.Dense(6, activation='softmax'))

    return model
```



شکل 12-1. تغییرات accuracy و loss در هنگام آموزش مدل LSTM

مقایسه و LSTM و CNN

CNN شبکه

نقاط قوت:

1. استخراج ویژگی‌های محلی: CNN‌ها به خوبی می‌توانند الگوهای محلی و ویژگی‌های فضایی در داده‌ها را شناسایی کنند. به همین دلیل در پردازش تصویر بسیار موثر هستند.
2. مقیاس‌پذیری: از آنجا که فیلترهای همگشتی پارامترهای کمتری نسبت به لایه‌های کاملاً متصل دارند، مدل‌های CNN مقیاس‌پذیرتر و کارآمدتر هستند.
3. تبادلات کمتر: CNN‌ها به دلیل استفاده از فیلترهای اشتراکی و pooling layers، به تبادلات کمتری برای یادگیری نیاز دارند.

نقاط ضعف:

1. عدم درک توالی زمانی: CNN‌ها به خوبی قادر به درک و حفظ روابط زمانی و ترتیبی نیستند.
2. سختی در پردازش داده‌های ترتیبی: برای داده‌هایی که دارای وابستگی‌های زمانی یا ترتیبی هستند، مانند سری‌های زمانی یا متن، CNN‌ها به تنها‌یی کارآمد نیستند.

LSTM شبکه

نقاط قوت:

1. حفظ اطلاعات طولانی مدت: LSTM‌ها برای حفظ وابستگی‌های طولانی مدت در داده‌ها طراحی شده‌اند. این ویژگی آن‌ها را برای پردازش داده‌های ترتیبی و سری‌های زمانی بسیار مناسب می‌سازد.
2. مدیریت گرادیان‌های از بین رفته: LSTM‌ها مشکل ناپدید شدن گرادیان را که در شبکه‌های عصبی عادی رخ می‌دهد، کاهش می‌دهند.

نقاط ضعف:

1. زمان آموزش طولانی‌تر: به دلیل پیچیدگی بیشتر در ساختار داخلی خود، LSTM‌ها به زمان و منابع محاسباتی بیشتری برای آموزش نیاز دارند.
2. کمتر کارآمد برای ویژگی‌های محلی: در مقایسه با CNN‌ها، LSTM‌ها برای استخراج ویژگی‌های محلی و فضایی کمتر مؤثر هستند.

LSTM و CNN

1. CNN برای استخراج ویژگی‌های محلی: ابتدا از لایه‌های CNN برای استخراج ویژگی‌های محلی و فضایی از داده‌ها (مانند تصاویر یا داده‌های ترتیبی) استفاده می‌شود.
2. LSTM برای تحلیل وابستگی‌های زمانی: سپس ویژگی‌های استخراج شده توسط CNN به لایه‌های LSTM داده می‌شود تا روابط زمانی و ترتیبی در داده‌ها تحلیل و مدل‌سازی شوند.

مثال کاربردی:

- **ویدئوها:** در تحلیل ویدئوها، CNN برای استخراج ویژگی‌های محلی از هر فریم و LSTM برای مدل‌سازی وابستگی‌های زمانی بین فریم‌ها استفاده می‌شود.
- **تحلیل متن و گفتار:** ابتدا CNN می‌تواند ویژگی‌های کلیدی از قطعات متن یا گفتار استخراج کند و سپس LSTM می‌تواند روابط ترتیبی و زمانی را تحلیل کند.

5-1. ارزیابی

در ادامه سه مدل گفته شده را با استفاده از پارامترهای بهینه آموزش دادیم و عملکرد مدل را بر روی داده‌های آزمون را در ادامه گزارش می‌کنیم:

1. مدل CNN-LSTM

Metric	Value		
Accuracy	0.872		
Class	Precision	Recall	F1-score
Class 0	0.8751	0.8554	0.8651
Class 1	0.9354	0.8066	0.8662
Class 2	0.9128	0.8722	0.892
Class 3	0.9083	0.8405	0.8731
Class 4	0.8139	0.9105	0.8595
Class 5	0.9103	0.8703	0.8898
Average Type	Precision	Recall	F1-score
Macro average	0.8926	0.8592	0.8743
Micro average	0.872	0.872	0.872
Weighted average	0.8753	0.872	0.8724

CNN مدل .2

Metric	Value		
Accuracy	0.8569		
Class	Precision	Recall	F1-score
Class 0	0.7824	0.8569	0.8179
Class 1	0.8905	0.8502	0.8699
Class 2	0.9138	0.8453	0.8782
Class 3	0.886	0.8301	0.8571
Class 4	0.8285	0.8962	0.861
Class 5	0.9428	0.8266	0.8809
Average Type	Precision	Recall	F1-score
Macro average	0.874	0.8509	0.8608
Micro average	0.8569	0.8569	0.8569
Weighted average	0.8612	0.8569	0.8575

LSTM مدل .3

Metric	Value
Accuracy	0.8781

Class	Precision	Recall	F1-score
Class 0	0.8867	0.8562	0.8712
Class 1	0.9094	0.9094	0.9094
Class 2	0.8991	0.8774	0.8881
Class 3	0.8437	0.9014	0.8716
Class 4	0.8888	0.871	0.8798
Class 5	0.8856	0.8793	0.8824
Average Type	Precision	Recall	F1-score
Macro average	0.8855	0.8825	0.8838
Micro average	0.8781	0.8781	0.8781
Weighted average	0.8788	0.8781	0.8781

Micro Averaging

در روش micro averaging، ابتدا تمام پیش‌بینی‌ها و کلاس‌ها به صورت یک مجموعه یکپارچه در نظر گرفته می‌شوند. سپس معیارهای ارزیابی بر اساس این مجموعه کلی محاسبه می‌شوند. این روش به این صورت عمل می‌کند که تعداد TP، TN، FP، FN ها برای تمام کلاس‌ها با هم جمع می‌شوند و سپس این مقادیر برای محاسبه معیارها به کار می‌روند.

روش micro averaging به کلاس‌هایی که تعداد نمونه‌های بیشتری دارند بیشتر وزن می‌دهد. بنابراین، این روش به شدت تحت تأثیر کلاس‌های با تعداد نمونه‌های زیاد است و ممکن است عملکرد مدل روی کلاس‌های با نمونه‌های کمتر را به خوبی نمایش ندهد.

Macro Averaging

در روش macro averaging، ابتدا معیارهای ارزیابی برای هر کلاس به صورت جداگانه محاسبه می‌شوند. سپس میانگین این مقادیر برای تمامی کلاس‌ها محاسبه می‌شود. به عبارت دیگر، برای هر کلاس یک F1-Score، Accuracy، Recall، Precision محاسبه شده و سپس این مقادیر با هم میانگین‌گیری می‌شوند.

روش macro averaging به هر کلاس وزن یکسانی می‌دهد، بدون توجه به تعداد نمونه‌های هر کلاس. این روش می‌تواند مشکلات ناشی از کلاس‌های نادر را برجسته کند، زیرا هر کلاس به صورت مستقل و برابر در محاسبه میانگین لحاظ می‌شود.

Weighted Averaging

در روش weighted averaging، ابتدا معیارهای ارزیابی برای هر کلاس به صورت جداگانه محاسبه می‌شوند، مشابه روش macro averaging. اما میانگین این معیارها بر اساس وزن هر کلاس محاسبه می‌شود، که این وزن معمولاً برابر با تعداد نمونه‌های هر کلاس است.

روش weighted averaging توازنی بین دو روش micro و macro averaging ایجاد می‌کند. به این صورت که معیارها بر اساس اهمیت (تعداد نمونه‌ها) هر کلاس میانگین‌گیری می‌شوند. این روش می‌تواند عملکرد کلی مدل را با در نظر گرفتن تعداد نمونه‌ها بهتر نمایش دهد، در حالی که هنوز به کلاس‌های نادر اهمیت می‌دهد.

6-1. امتیازی

در ابتدا با استفاده از کیسه کلمات²²، برای هر جمله یک بردار ساخته شده است. سپس از پنج الگوریتم یادگیری ماشین کلاسیک برای یادگیری مدل استفاده کردیم.

ابتدا روش کیسه کلمات را مختصررا توضیح می‌دهیم:

روش Bag of Words یک تکنیک ساده و رایج در پردازش زبان طبیعی و یادگیری ماشین است که برای نمایش متن استفاده می‌شود. در این روش، هر متن به عنوان مجموعه‌ای از کلمات (یا توکن‌ها) در نظر گرفته می‌شود بدون در نظر گرفتن ترتیب آن‌ها. به طور خلاصه، مراحل این روش به شرح زیر است:

1. **توکن‌سازی (Tokenization)**: متن به کلمات جداگانه (توکن‌ها) تقسیم می‌شود.
2. **ساخت واژگان (Vocabulary Creation)**: لیستی از تمام کلمات منحصر به فرد (واژگان) موجود در مجموعه متون تهیه می‌شود.
3. **ایجاد بردار ویژگی‌ها (Feature Vector Creation)**: هر متن به برداری تبدیل می‌شود که طول آن برابر با تعداد کلمات در واژگان است. هر عنصر از این بردار تعداد دفعات تکرار هر کلمه در آن متن را نشان می‌دهد.

این روش ساده اما موثر است، اما یکی از نقاط ضعف آن نادیده گرفتن ترتیب کلمات و روابط نحوی بین آن‌هاست.

به علت زیاد شدن ابعاد فضایی بردارهای ویژگی در روش کیسه کلمات، از همه‌ی داده‌ها استفاده نکردیم و از 20,000 داده برای این کار استفاده کردیم. (حدود 18 درصد کل داده‌ها)

²² Bag of Words

در ادامه با 5 روش به کاربرده شده آشنا می‌شویم:

:Gaussian Naive Bayes .1

یک الگوریتم ساده و سریع طبقه‌بندی که بر اساس قضیه بیز و فرض استقلال بین ویژگی‌ها کار می‌کند. در Gaussian Naive Bayes فرض می‌شود که داده‌ها از توزیع نرمال (گوسی) پیروی می‌کنند.

Classifier	Accuracy
Gaussian Naive Bayes	0.35025

Classification Report for Gaussian Naive Bayes:

Class	Precision	Recall	F1-score	Support
0	0.371179	0.35124	0.360934	726
1	0.25	0.0344828	0.0606061	29
2	0.327343	0.430017	0.37172	593
3	0.450723	0.35	0.394026	980
4	0.451398	0.281095	0.346449	1206
5	0.204322	0.446352	0.280323	466

Average Metrics:

Classifier	Gaussian Naive Bayes
Macro Precision	0.342494
Macro Recall	0.315531
Macro F1-score	0.302343

Weighted Precision	0.388037
Weighted Recall	0.35025
Weighted F1-score	0.354705

:Decision Tree .2

یک مدل طبقه‌بندی که از ساختار درختی برای اتخاذ تصمیمات استفاده می‌کند.
هر گره داخلی یک ویژگی را تست می‌کند، هر شاخه نشان‌دهنده نتیجه آن تست
و هر برگ یک کلاس یا ارزش خروجی است.

Classifier	Accuracy
Decision Tree	0.872

Classification Report for Decision Tree:

Class	Precision	Recall	F1-score	Support
0	0.872253	0.874656	0.873453	726
1	0.84375	0.931034	0.885246	29
2	0.886598	0.870152	0.878298	593
3	0.886831	0.879592	0.883197	980
4	0.844517	0.855721	0.850082	1206
5	0.896552	0.892704	0.894624	466

Average Metrics:

Classifier	Decision Tree
------------	---------------

Macro Precision	0.87175
Macro Recall	0.883976
Macro F1-score	0.877483
Weighted Precision	0.872213
Weighted Recall	0.872
Weighted F1-score	0.872064

:Gradient Boosting .3

یک روش قدرتمند و انعطاف‌پذیر برای طبقه‌بندی که مدل درخت تصمیم را به صورت تکراری ترکیب می‌کند تا مدل نهایی قوی‌تر شود. هر مدل جدید خطاهای مدل‌های قبلی را تصحیح می‌کند.

نکته: اجرای این مدل به بیش از 3 ساعت به طول انجامید ولی به اتمام نرسید!

:Random Forest .4

یک الگوریتم ensemble که چندین درخت تصمیم را با هم ترکیب می‌کند. هر درخت با استفاده از نمونه‌گیری تصادفی از داده‌ها و ویژگی‌ها آموزش داده می‌شود. نتایج نهایی با توجه به نتایج درختان حاصل می‌شود.

Classifier	Accuracy
Random Forest	0.89825

Classification Report for Random Forest:

Class	Precision	Recall	F1-score	Support
0	0.938596	0.884298	0.910638	726

1	0.961538	0.862069	0.909091	29
2	0.942446	0.883642	0.912097	593
3	0.888	0.906122	0.89697	980
4	0.846978	0.91791	0.881019	1206
5	0.953162	0.873391	0.911534	466

Average Metrics:

Classifier	Random Forest
Macro Precision	0.921787
Macro Recall	0.887905
Macro F1-score	0.903558
Weighted Precision	0.901011
Weighted Recall	0.89825
Weighted F1-score	0.898669

:Logistic Regression .5

یک مدل خطی برای طبقه‌بندی که احتمال وابستگی به کلاس‌ها را بر اساس یک تابع sigmoid محاسبه می‌کند.

Classifier	Accuracy
Logistic Regression	0.897

Classification Report for Logistic Regression:

Class	Precision	Recall	F1-score	Support
0	0.894882	0.891185	0.89303	726
1	0.964286	0.931034	0.947368	29
2	0.917832	0.885329	0.901288	593
3	0.905057	0.894898	0.899949	980
4	0.866086	0.906302	0.885737	1206
5	0.939462	0.899142	0.91886	466

Average Metrics:

Classifier	Logistic Regression
Macro Precision	0.914601
Macro Recall	0.901315
Macro F1-score	0.907705
Weighted Precision	0.897792
Weighted Recall	0.897
Weighted F1-score	0.897154

پرسش 2 - سامانه‌های سایبرفیزیکی²³: نگهداری هوشمند²⁴

سامانه‌های سایبرفیزیکی که در سال 2006 توسط دانشگاه برکلی معرفی شدند، سامانه‌هایی هستند که از حسگرها و کانال‌های ارتباطی برای جمعآوری اطلاعات از محیط فیزیکی استفاده می‌کنند، آن‌ها را تجزیه و تحلیل می‌کنند و سپس محیط فیزیکی را تحت تاثیر قرار می‌دهند. با پیشرفت فناوری‌های اینترنت اشیا، این سامانه‌ها به دیجیتالیزه شدن صنعت کمک کرده‌اند. این سامانه‌ها امکاناتی را برای فرآیند تولید با قابلیت اطمینان، در دسترس بودن، قابلیت نگهداری و ایمنی بال فراهم می‌کنند. اما از طرف دیگر، ارزیابی سلامت این سامانه‌ها پیچیده و چالش‌برانگیز است. در این تمرین، ما از مدل پیشنهادی [این مقاله](#) برای تشخیص و رگرسیون عمر مفید باقیمانده استفاده می‌کنیم. این مقاله یک مدل یادگیری عمیق CNN-LSTM را معرفی کرده است و آزمایش‌هایی را بر روی مجموعه داده‌های [NASA's C-MAPSS](#) انجام داده‌ایم.

1-2. پیش‌پردازش داده‌ها

ابتدا توضیح مختصری در مورد دیتاست می‌دهیم:

مجموعه داده‌ها شامل چندین سری زمانی چند متغیره است. هر مجموعه داده به زیرمجموعه‌های آموزش و آزمون تقسیم می‌شود. هر سری زمانی از یک موتور متفاوت است، یعنی می‌توان داده‌ها را از یک دسته از موتورهای یکسان در نظر گرفت. هر موتور با درجات مختلفی از تنظیمات اولیه و تغییرات ساخت و ساز که برای ما ناشناخته است، شروع می‌شود. این تنظیمات و تغییرات به عنوان حالت نرمال در نظر گرفته می‌شود، یعنی به عنوان یک شرایط خرابی در نظر گرفته نمی‌شود. سه تنظیم عملیاتی²⁵ وجود دارد که تاثیر قابل توجهی بر عملکرد موتور دارد. این تنظیمات نیز در داده‌ها گنجانده شده‌اند. باید توجه داشته باشیم که داده‌ها با نویز سنسور آلوده شده‌اند.

موتور در ابتدای هر سری زمانی به طور نرمال کار می‌کند و در یک نقطه در طول سری، یک خرابی توسعه می‌یابد. در مجموعه آموزش، خرابی تا زمانی که سیستم خراب شود، افزایش می‌یابد. در مجموعه آزمون، سری زمانی کمی قبل از شکست سیستم پایان می‌یابد. هدف از این

²³ Cyber-Physical Systems (CPSs)

²⁴ Intelligent Maintenance

²⁵ Operational Setting

رقابت پیش‌بینی تعداد چرخه‌های عملیاتی باقی‌مانده قبل از شکست در مجموعه آزمون است، یعنی تعداد چرخه‌های عملیاتی پس از آخرین چرخه که موتور همچنان کار خواهد کرد. همچنین مقادیر واقعی عمر مفید باقی‌مانده (RUL) برای داده‌های آزمون ارائه شده است.

داده‌ها به صورت یک فایل با 26 ستون از اعداد، جدا شده با فاصله، ارائه شده‌اند. هر ردیف مقادیر ثبت شده توسط سنسورها در طول یک چرخه عملیاتی است، هر ستون یک متغیر متفاوت است. ستون‌ها مربوط به:

(1) شماره واحد موتور (نام موتور)

(2) زمان فعال بودن این واحد موتور

(3-5) تنظیم عملیاتی 1-3

(6-26) اندازه‌گیری سنسور 1-26

سه مجموعه داده نیز داریم که مقادیر برای انواع مدل موتورها در آن‌ها ثبت شده که به شکل زیر می‌باشد:

(1) مجموعه داده: FD001

واحدهای آموزش: 100

واحدهای آزمون: 100

شرایط: یک (سطح دریا)

حالت‌های خرابی: یک (تخربی HPC)

(2) مجموعه داده: FD002

واحدهای آموزش: 260

واحدهای آزمون: 259

شرایط: شش

حالت‌های خرابی: یک (تخربی HPC)

(3) مجموعه داده: FD003

واحدهای آموزش: 100

واحدهای آزمون: 100

شرایط: یک (سطح دریا)

حالتهای خرابی: دو (تخرب HPC، تخریب فن)

FD004 (4) مجموعه داده

مسیرهای آموزش: 248

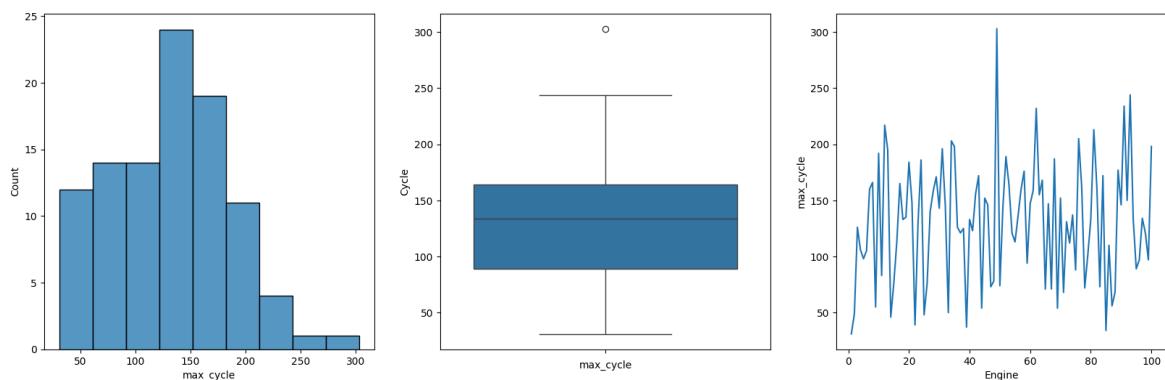
مسیرهای آزمون: 249

شرایط: شش

حالتهای خرابی: دو (تخرب HPC، تخریب فن)

```
def load_data(file_path: str, col_names: list) -> pd.DataFrame:
    data = pd.read_csv(file_path, sep="\s+", header=None,
names=col_names)
    print(f"Data loaded successfully from {file_path}")
    return data

df_train = load_data(TRAIN_DIR, col_names)
df_test = load_data(TEST_DIR, col_names)
df_rul = load_data(RUL_DIR, rul_col)
df_rul[["Engine"]] = df_rul.index + 1
```

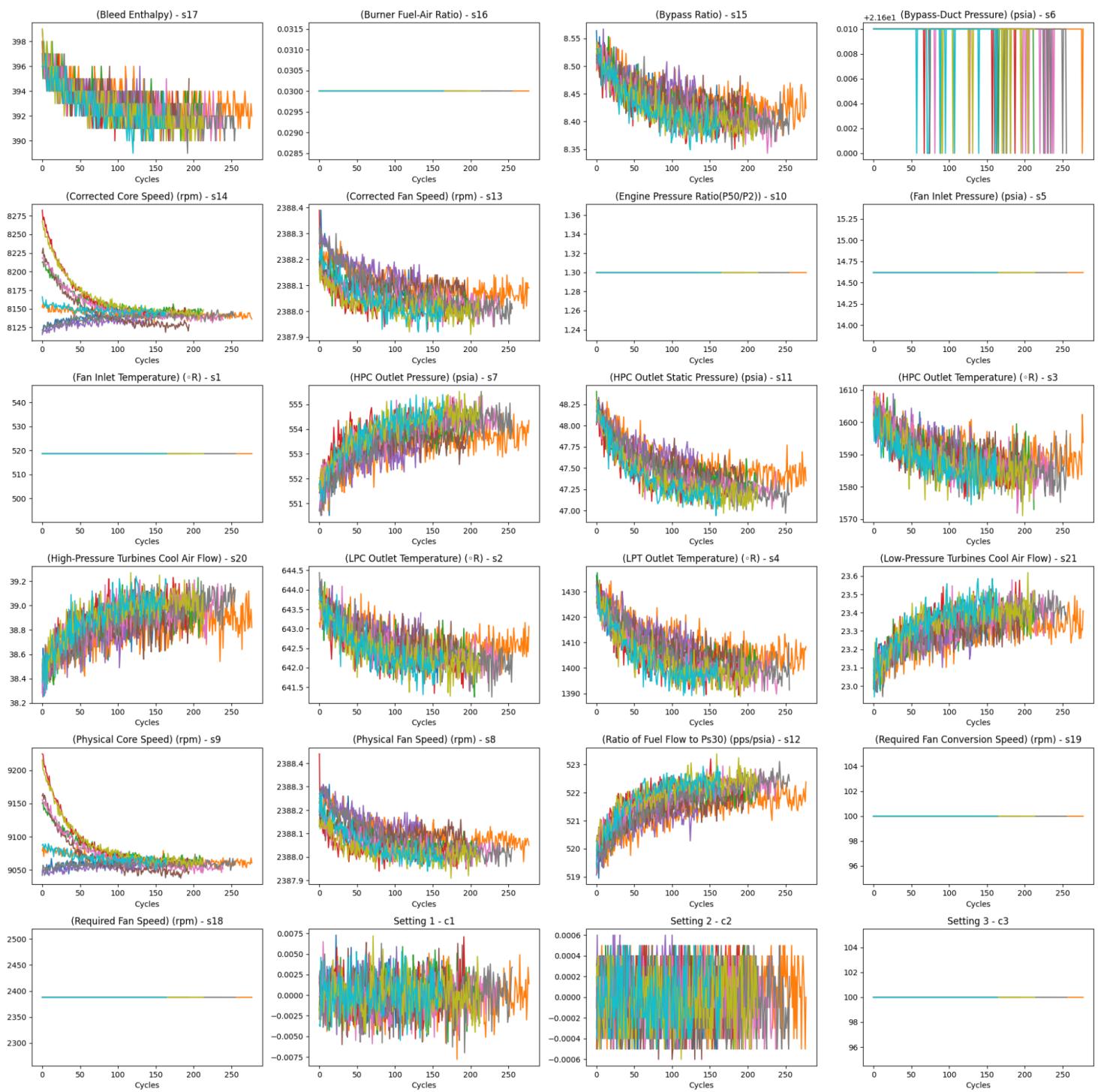


شكل 2-1. توزيع تعداد cycle-ها در دیتاست train

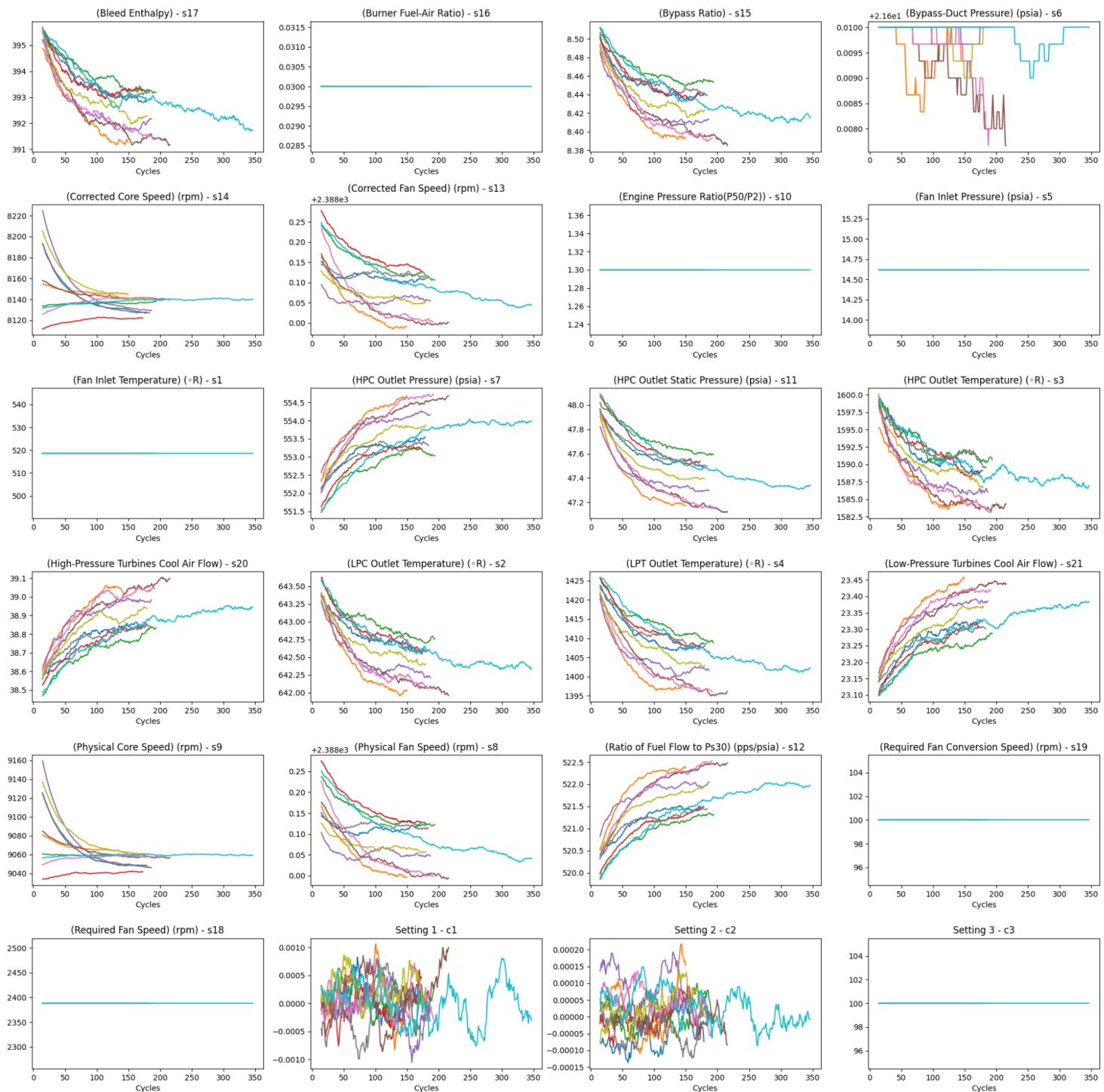
Data Selection .1

در مجموعه داده 3 تنظیم عملیاتی و اندازه‌گیری‌های واحد موتور از 21 سنسور وجود دارد که می‌توانند اطلاعات مهمی در مورد واحدهای موتور را فراهم کنند. با این حال، از شکل 2 و شکل 3 مشاهده می‌شود که بخش‌هایی از نمونه‌های حسگر تغییرات قابل توجهی را نشان نمی‌دهند. این داده‌ها نمی‌توانند اطلاعات مفیدی درباره RUL²⁶ ارائه دهند و پیچیدگی شبکه‌های عصبی را افزایش می‌دهند. بنابراین، این ویژگی‌ها از مجموعه داده حذف خواهند شد که شاخص‌های آن‌ها s16، s19، s10، c3، s1، s5، و s19 است. همچنین چند ویژگی مانند s6 با اینکه در طول زمان ثابت نیست اما حرکتی تقریباً نامشخص دارد و باعث پیچیدگی و عدم آموزش درست مدل می‌شود پس می‌توان آن را نیز حذف کرد.

²⁶ Remaining Useful Life



شکل 2-2. اعداد ثبت شده توسط سنسورها با نویز در طول زمان برای 10 موتور



شكل 2-3. اعداد ثبت شده توسط سنسورها بدون نویز (میانگین در یک window) در طول زمان برای 10 موتور

Data Labeling .2

حال باید برای داده‌ها برای classification و regression تعدادی label درست کنیم که برای بدست آوردن مقدار RUL به شکل زیر عمل کردیم:

```
def add_remaining_useful_life(df: pd.DataFrame,
cycles_after_last_cycle: pd.Series = None) -> pd.DataFrame:
    if cycles_after_last_cycle is not None:
        max_cycle = df.groupby("Engine")["Cycle"].transform("max")
        df = df.merge(cycles_after_last_cycle, on="Engine",
how="left")
        df["RUL"] = df["RUL"] + (max_cycle - df["Cycle"])
    else:
        max_cycle = df.groupby("Engine")["Cycle"].transform("max")
        df["RUL"] = max_cycle - df["Cycle"]
    return df
```

```
df_train = add_remaining_useful_life(df_train)
df_test = add_remaining_useful_life(df_test, df_rul)
```

همچنین با توجه به مقاله برای پیش‌بینی بهتر مقدار RUL یک UPPER_LIMIT را به اندازه 130 برای داده‌ها در نظر گرفتیم و تمامی مقدارهای بیشتر از 130 را با 130 جایگزین کردیم که مدل سریع‌تر همگرا شود.

```
RUL_UPPER_BOUND = 130
df_train["RUL"] = df_train["RUL"].clip(upper=RUL_UPPER_BOUND)
df_test["RUL"] = df_test["RUL"].clip(upper=RUL_UPPER_BOUND)
```

سپس برای ساخت label-ها برای classification از این رابطه استفاده کردیم:

```
def add_class_label(df: pd.DataFrame, threshold: int =
WINDOW_SIZE) -> pd.DataFrame:
    df["label"] = df["RUL"].apply(lambda x: 1 if x <= threshold
else 0)
    return df
df_train = add_class_label(df_train)
df_test = add_class_label(df_test)
```

Data Normalization .3

در این بخش مثل مقاله از **MinMaxScaler** استفاده شده.

```
scaler = MinMaxScaler()
df_train[df_train.columns.difference(["Engine", "Cycle", "RUL",
"label"])] = (
    scaler.fit_transform(
        df_train[df_train.columns.difference(["Engine", "Cycle",
"RUL", "label"])]
    )
)
df_test[df_test.columns.difference(["Engine", "Cycle", "RUL",
"label"])] = (
    scaler.transform(
        df_test[df_test.columns.difference(["Engine", "Cycle",
"RUL", "label"])]
    )
)
```

Time Windowing .4

حال طبق مقاله به ساخت تعدادی window می‌پردازیم و با کمک آن‌ها به آموزش و بررسی مدل می‌پردازیم. طول window به عنوان N_L و stride به عنوان k بیان می‌شود. با توجه به اینکه stride کوتاه می‌توانند تعداد نمونه‌های آموزشی را افزایش دهند و این ممکن است خطر overfitting را کاهش دهد، منطقی است که k را برابر با ۱ تنظیم کنیم. نمونه‌های داده در یک پنجره زمانی به واحد موتور یکسانی تعلق دارند. کلاس هدف و RUL آخرین نقطه داده به عنوان برچسب هدف برای هر پنجره زمانی استفاده می‌شود. مقدار پارامتر N_L باید کوچکتر از طول داده‌های ضبط (طول مسیرهای ضبط شده) شده باشد. علاوه بر این، هرچه مقدار N_L بیشتر باشد، مدل‌ها توانایی بیشتری برای نگاه به گذشته در داده‌های تاریخچه خواهند داشت. با این حال، این امر منجر به افزایش زمان آموزش خواهد شد. بنابراین، برای هر مجموعه آزمون در مطالعه موردی، ترجیح داده می‌شود که مقادیر N_L به اندازه کافی بزرگ باشد تا از اطلاعات تاریخچه بهره‌مند شوند، اما باید کوچکتر از حداقل طول مسیرهای ضبط شده باشد. با توجه به این که حداقل طول ضبط شده در مجموعه داده C-MAPSS FD001 برابر با ۳۱ چرخه است، بنابراین مقدار N_L برابر با ۳۰ چرخه تنظیم شده است.

```

def time_window_data(
    df: pd.DataFrame,
    window_size: int = WINDOW_SIZE,
    stride: int = 1
) -> tuple[np.ndarray, np.ndarray, np.ndarray]:
    data = []
    rules = []
    labels = []
    for engine in df["Engine"].unique():
        df_engine = df[df["Engine"] == engine].copy()
        for i in range(window_size, df_engine.shape[0] + 1,
                      stride):
            data.append(df_engine.iloc[i - window_size :
                                         i].drop(["Engine", "Cycle", "RUL", "label"], axis=1).values)
            rules.append(df_engine.iloc[i - 1]["RUL"])
            labels.append(df_engine.iloc[i - 1]["label"])

    return np.array(data), np.array(rules), np.array(labels)

```

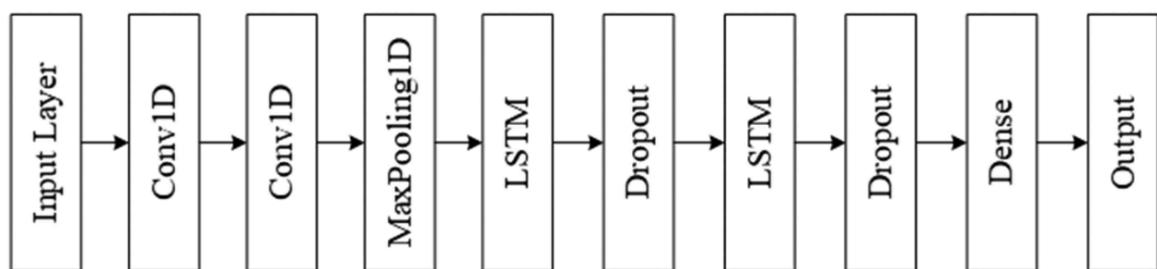
```

X_train, y_train, y_train_label = time_window_data(df_train)
X_test, y_test, y_test_label = time_window_data(df_test)

```

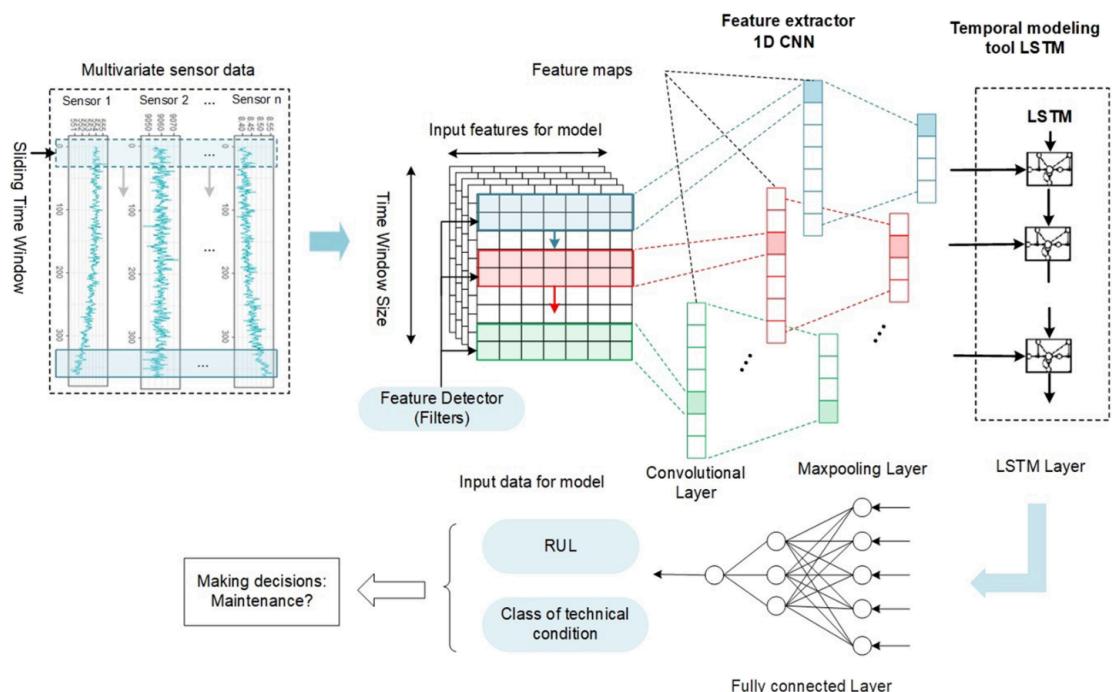
2-2. مدلسازی و ارزیابی

در اینجا مدل پیشنهادی توسط مقاله را پیاده‌سازی می‌کنیم. مدل پیشنهادی ساختاری شبیه به شکل زیر دارد:



شکل 2-4. معماری مدل پیشنهادی

که اینگونه عمل می‌کند که ابتدا شبکه CNN ویژگی و feature-ها را استخراج می‌کند و به شبکه LSTM می‌دهد تا با توجه به گذشته به درستی تصمیم بگیرد. این دو شبکه در این حالت مکمل هم هستند، در حالی که فقط یکی از این دو استفاده شده باشد (که جلوتر پیاده‌سازی شده) می‌بینیم که با اینکه دقیق خوبی دارند اما استفاده از این دو که عیوب‌های یکدیگر را تکمیل کنند باعث می‌شود مدل دقیق‌تر داشته باشد. در شکل پایین می‌توانید معماری و عملکرد این مدل را دقیق‌تر ببینید:



شکل ۵-۲. معماری مدل پیشنهادی با جزییات بیشتر

برای همه لایه‌ها از ReLU به عنوان activation function استفاده شده و جزییات بیشتر مدل‌ها و تعداد نورون و بقیه پارامترها در کد قابل مشاهده می‌باشد.

Classification

در مرحله قبل کلاس هر time window مشخص شد پس الان میتوانیم با کمک آنها به طبقه‌بندی بپردازیم. معماری مدل را در شکل زیر می‌بینید:

```
def initialize_hybrid_CNN_LSTM_classification_model(
    input_shape: tuple,
    dropout_rate: float = 0.2,
    l2_regularizer: float = 0.01
) -> models.Sequential:
    model = Sequential()
    model.add(layers.Conv1D(64, 3, activation="relu",
input_shape=input_shape))
    model.add(layers.MaxPooling1D(2))
    model.add(layers.Conv1D(64, 3, activation="relu"))
    model.add(layers.MaxPooling1D(2))
    model.add(layers.LSTM(64, return_sequences=True))
    model.add(Dropout(dropout_rate))
    model.add(layers.LSTM(64, return_sequences=False))
    model.add(Dropout(dropout_rate))
    model.add(Dense(1, activation="sigmoid",
kernel_regularizer=regularizers.l2(l2_regularizer)))

    model.compile(optimizer=Adam(learning_rate=0.001),
loss="binary_crossentropy", metrics=["accuracy"])
    return model
```

Model: "sequential_11"

Layer (type)	Output Shape	Param #
<hr/>		
conv1d_9 (Conv1D)	(None, 28, 64)	3328
max_pooling1d_9 (MaxPooling 1D)	(None, 14, 64)	0
conv1d_10 (Conv1D)	(None, 12, 64)	12352

```

max_pooling1d_10 (MaxPooling1D)          0
                                         (None, 6, 64)

lstm_24 (LSTM)                         33024
                                         (None, 6, 64)

dropout_27 (Dropout)                   0
                                         (None, 6, 64)

lstm_25 (LSTM)                         33024
                                         (None, 64)

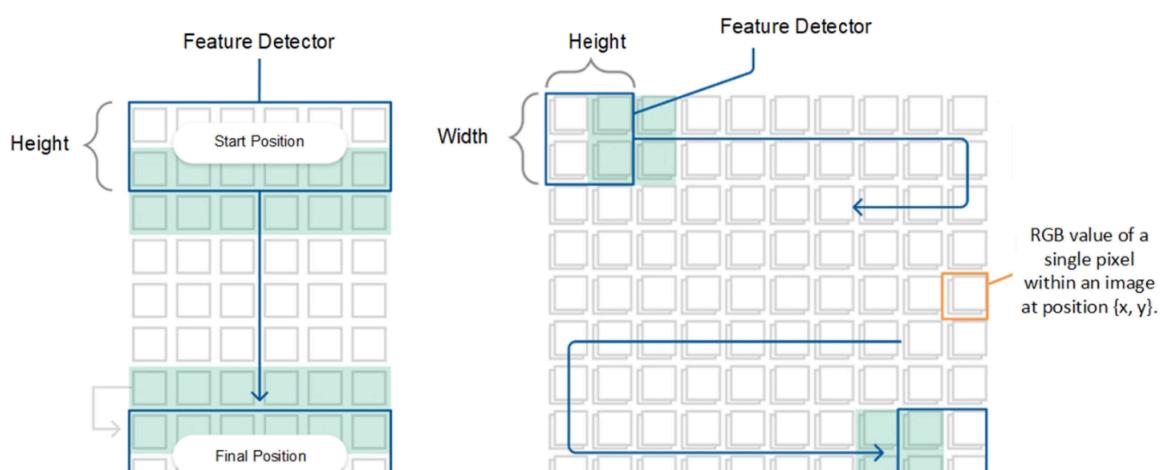
dropout_28 (Dropout)                   0
                                         (None, 64)

dense_14 (Dense)                      65
                                         (None, 1)

=====
=
Total params: 81,793
Trainable params: 81,793
Non-trainable params: 0

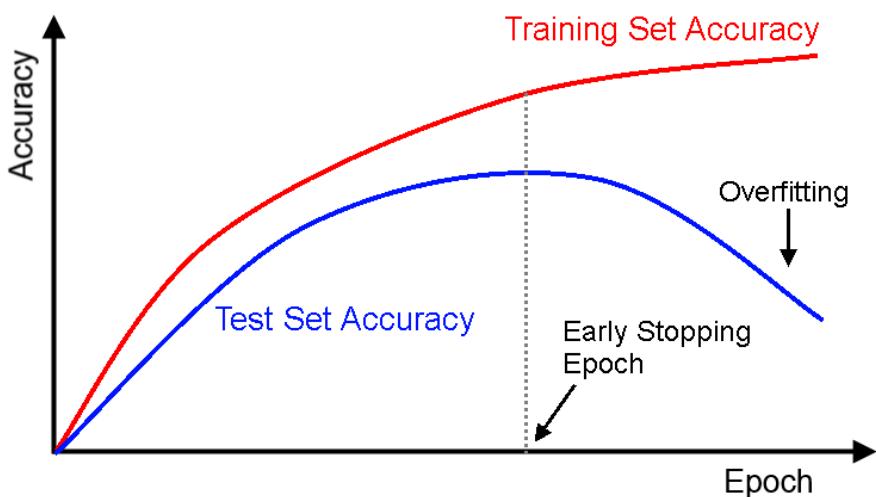
```

توجه کنید که برای جلوگیری از overfitting dropout از لایه‌های LSTM استفاده شده. در اینجا از binary cross entropy برای تابع loss استفاده شده. دلیل استفاده از Conv1D هم به خاطر این است که بتوانیم ویژگی‌ها را استخراج کنیم، تفاوت این لایه با Conv2D در شکل زیر نشان داده شده:



شکل 2-6. تفاوت لایه‌های Conv2D و Conv1D

حال مدل را با و بدون آموزش می‌دهیم. این روش یکی از محبوب‌ترین و همچنین موثرترین تکنیک‌ها برای جلوگیری از بیش‌برازش است. در فرآیند آموزش، ۲۰٪ از داده‌های آموزشی انتخاب می‌شود تا در پایان هر دوره آموزشی،تابع loss محاسبه شود. هنگامی که کاهش تابع loss متوقف شد، آموزش متوقف شده و از داده‌های آزمون برای محاسبه دقت نهایی مدل استفاده می‌شود. با این کار زمان آموزش کاهش می‌یابد و باعث می‌شود زمانی که مدل می‌خواهد شروع به overfitting کند (شکل زیر) آموزش متوقف شود.



شکل 2-6. نمودار تغییر loss در زمان آموزش

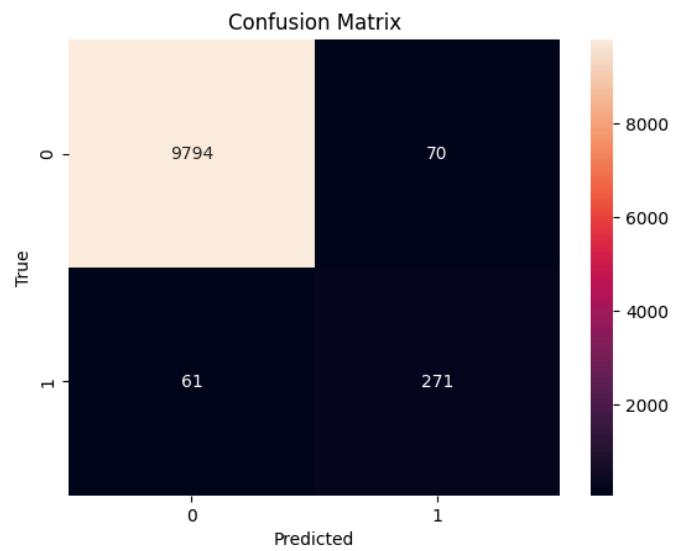
بدون early stopping

```
history_classification_hybrid_cnn_lstm =  
hybrid_cnn_lstm_classification_without_early_stopping_model.fit(  
    X_train,  
    y_train_label,  
    batch_size=64,  
    validation_data=(X_valid, y_valid_label),  
    epochs=100,  
    validation_split=0.2,  
    verbose=1,  
)
```

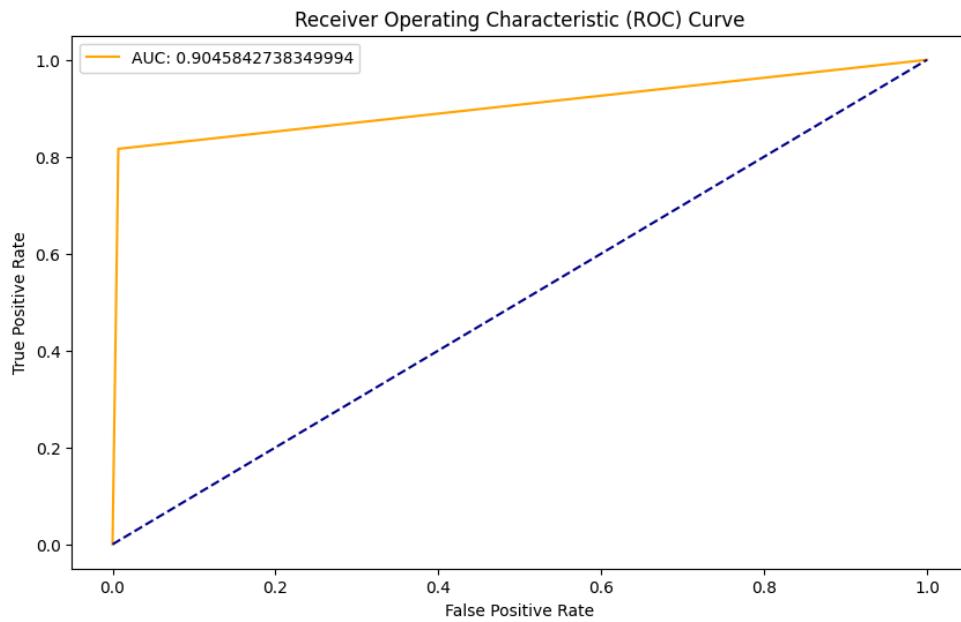
در این حالت مدل برای 100 ایپاک آموزش دید و نتایج به این شکل بود:

```
Epoch 100/100  
222/222 [=====] - 4s 17ms/step - loss:  
100.9667 - mean_squared_error: 98.5379 - val_loss: 106.7665 -  
val_mean_squared_error: 104.3446
```

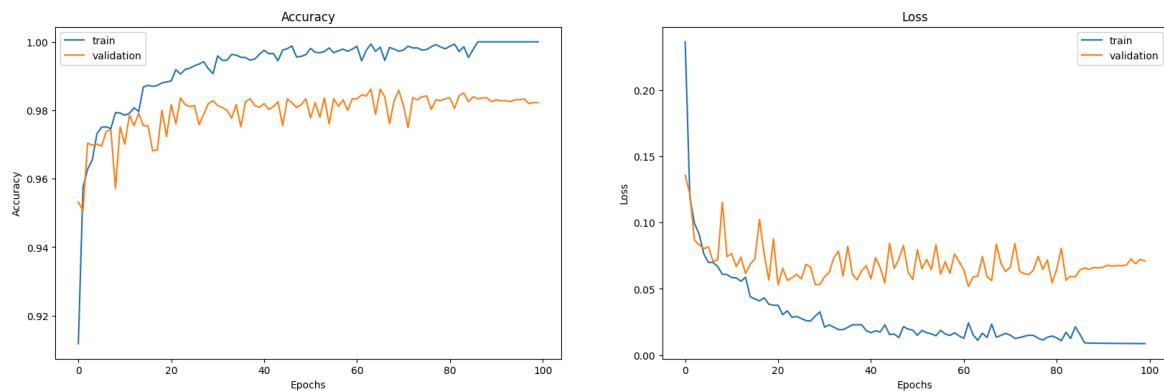
	precision	recall	f1-score	support
0.0	0.99	0.99	0.99	9864
1.0	0.79	0.82	0.81	332
accuracy			0.99	10196
macro avg	0.89	0.90	0.90	10196
weighted avg	0.99	0.99	0.99	10196



شکل 2-7. Confusion Matrix برای مدل هایبرید classification پیشنهادی بدون early stopping



شکل 2-8. نمودار ROC برای مدل هایبرید classification پیشنهادی بدون early stopping



شکل 2-9. نمودار آموزش برای مدل هایبرید classification پیشنهادی بدون early stopping

از نتایج و همچنین مشاهده نمودارها می‌توان نتیجه گرفت که مدل به خوبی آموزش دیده و با دقت خوبی می‌تواند طبقه‌بندی کند. دلیل اینکه در این حالت میزان کمی overfitting دیده می‌شود نیز استفاده از regularization l2 و dropout است که اگر حذف شوند و از رویکرد early stopping استفاده نشود مدل می‌تواند آموزش را به خوبی پیش‌بینی کند اما قابلیت تعمیم پذیری خوبی ندارد.

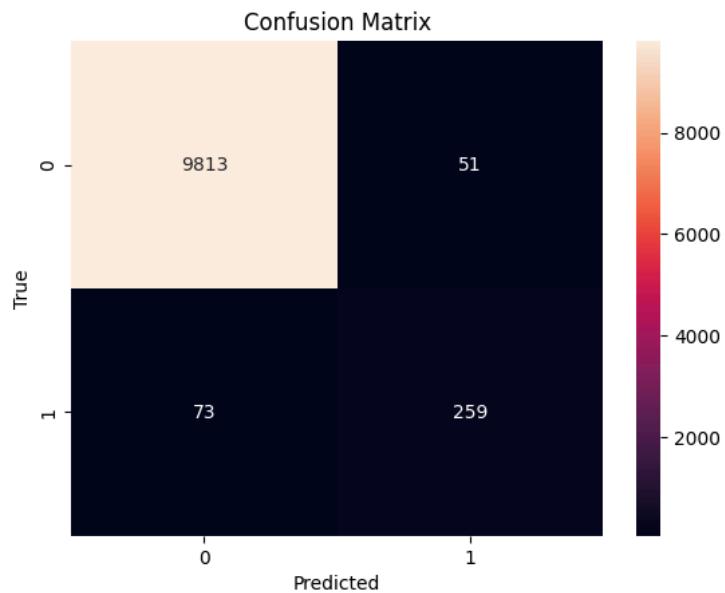
early stopping چی

```
history_classification_hybrid_cnn_lstm_with_early_stopping_model  
= hybrid_cnn_lstm_classification_with_early_stopping_model.fit(  
    X_train,  
    y_train_label,  
    batch_size=64,  
    validation_data=(X_valid, y_valid_label),  
    epochs=100,  
    validation_split=0.2,  
    callbacks=[EarlyStopping(patience=5)],  
    verbose=1,  
)
```

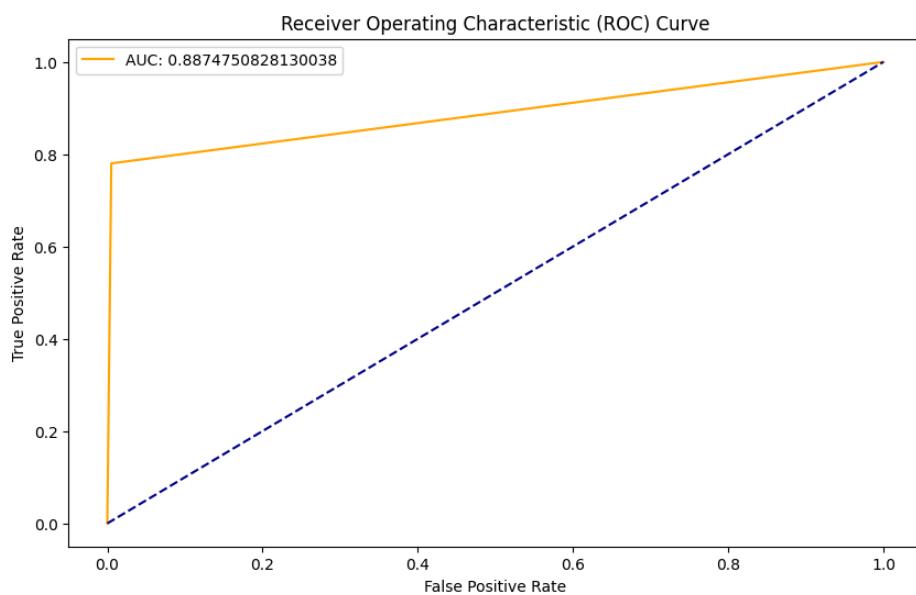
در این حالت مدل برای 28 ایپاک آموزش دید و نتایج به این شکل بود:

```
Epoch 28/100  
222/222 [=====] - 3s 12ms/step - loss:  
0.0327 - accuracy: 0.9910 - val_loss: 0.0555 - val_accuracy:  
0.9820
```

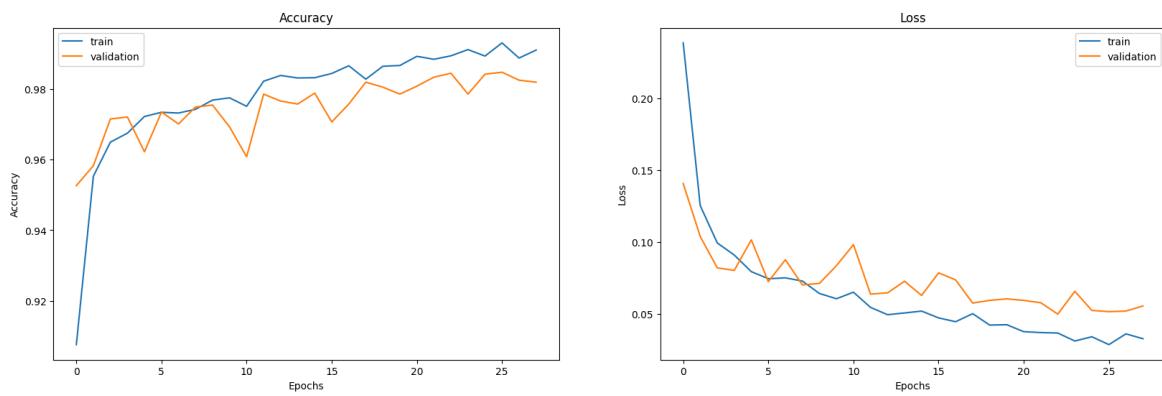
	precision	recall	f1-score	support
0.0	0.99	0.99	0.99	9864
1.0	0.84	0.78	0.81	332
accuracy			0.99	10196
macro avg	0.91	0.89	0.90	10196
weighted avg	0.99	0.99	0.99	10196



شکل 2. Confusion Matrix برای مدل هایبرید classification پیشنهادی با stopping



شکل 2. ROC نمودار برای مدل هایبرید classification پیشنهادی بدون stopping



شکل 2-12. نمودار آموزش برای مدل هایبرید classification پیشنهادی بدون early stopping

همانگونه که از نمودارهای شکل 2-12 می‌توان دید مدل بسیار کمتر overfitting انجام می‌دهد و دلیل اینکه دقت و auc به مقدار ناچیزی پایین‌تر است این است که داده‌ها پیچیدگی زیادی داشتند و مدل در چند ایپاک نتوانسته زیاد پیشرفت کند و این به اشتباه با نقطه بهینه اشتباه گرفته شده، می‌توان مدل را با patience بیشتری اجرا کرد که تاثیر آن بهتر دیده شود. دلیل اینکه ما استفاده نکردیم این است که در 100 ایپاک مدل به طور کامل شروع به overfitting نمی‌کند (به دلیل استفاده از دیگر روش‌ها که بالاتر گفته شد) و به خاطر همین استفاده از early stopping با اینکه زمان آموزش را به طور قابل توجهی کاهش می‌دهد اما ممکن است به مقدار کمی دقت را هم کاهش دهد. در تعداد ایپاک بالاتر استفاده از این روش بسیار بهتر نتیجه می‌دهد. توجه شود که مدل هنوز با دقت بسیار قابل قبولی طبقه‌بندی می‌کند.

Regression

مقدار RUL برای هر داده را در بخش‌های قبل بدست آوردیم، حال با کمک آن‌ها مسئله Regression را به مانند classification با دو روش و همچنین اینجا با در نظر گرفتن همه window-ها و تنها آخرین window مدل را تست می‌کنیم و با هم مقایسه می‌کنیم.

```
def initialize_hybrid_CNN_LSTM_regression_model(  
    input_shape: tuple,  
    dropout_rate: float = 0.2,  
    l2_regularizer: float = 0.01  
) -> models.Sequential:  
    model = Sequential()  
    model.add(layers.Conv1D(64, 3, activation="relu",  
    input_shape=input_shape))  
    model.add(layers.MaxPooling1D(2))  
    model.add(layers.Conv1D(64, 3, activation="relu"))  
    model.add(layers.MaxPooling1D(2))  
    model.add(layers.LSTM(64, return_sequences=True))  
    model.add(Dropout(dropout_rate))  
    model.add(layers.LSTM(64, return_sequences=False))  
    model.add(Dropout(dropout_rate))  
    model.add(Dense(1, activation="linear",  
    kernel_regularizer=regularizers.l2(l2_regularizer)))  
  
    model.compile(optimizer=RMSprop(learning_rate=0.001),  
    loss="mean_squared_error", metrics=["mean_squared_error"])  
    return model
```

Model: "sequential_14"

Layer (type)	Output Shape	Param #
<hr/>		
conv1d_15 (Conv1D)	(None, 28, 64)	3328
max_pooling1d_15 (MaxPooling1D)	(None, 14, 64)	0

```

conv1d_16 (Conv1D)           (None, 12, 64)          12352
max_pooling1d_16 (MaxPooling1D) (None, 6, 64)         0
lstm_30 (LSTM)               (None, 6, 64)          33024
dropout_32 (Dropout)         (None, 6, 64)          0
lstm_31 (LSTM)               (None, 64)             33024
dropout_33 (Dropout)         (None, 64)             0
dense_16 (Dense)             (None, 1)              65
=====
=
Total params: 81,793
Trainable params: 81,793
Non-trainable params: 0
-----
-
```

early stopping بذوق

```

history_regression_hybrid_cnn_lstm =
hybrid_cnn_lstm_regression_without_early_stopping_model.fit(
    X_train,
    y_train,
    batch_size=64,
```

```

        validation_data=(X_valid, y_valid),
        epochs=100,
        validation_split=0.2,
        verbose=1,
)

```

نتایج این بخش که 100 ایپاک آموزش دید به شکل زیر بود:

```

Epoch 100/100
222/222 [=====] - 4s 17ms/step - loss:
100.9667 - mean_squared_error: 98.5379 - val_loss: 106.7665 -
val_mean_squared_error: 104.3446

```

با توجه به اینکه دقت این حالت با حالت بعدی یکسان بود صرفا نتایج یک بخش نشان داده شده.

early stopping با

```

history_regression_hybrid_cnn_lstm_with_early_stopping =
hybrid_cnn_lstm_regression_with_early_stopping_model.fit(
    X_train,
    y_train,
    batch_size=64,
    validation_data=(X_valid, y_valid),
    epochs=100,
    validation_split=0.2,
    callbacks=[EarlyStopping(patience=15)],
    verbose=1,
)

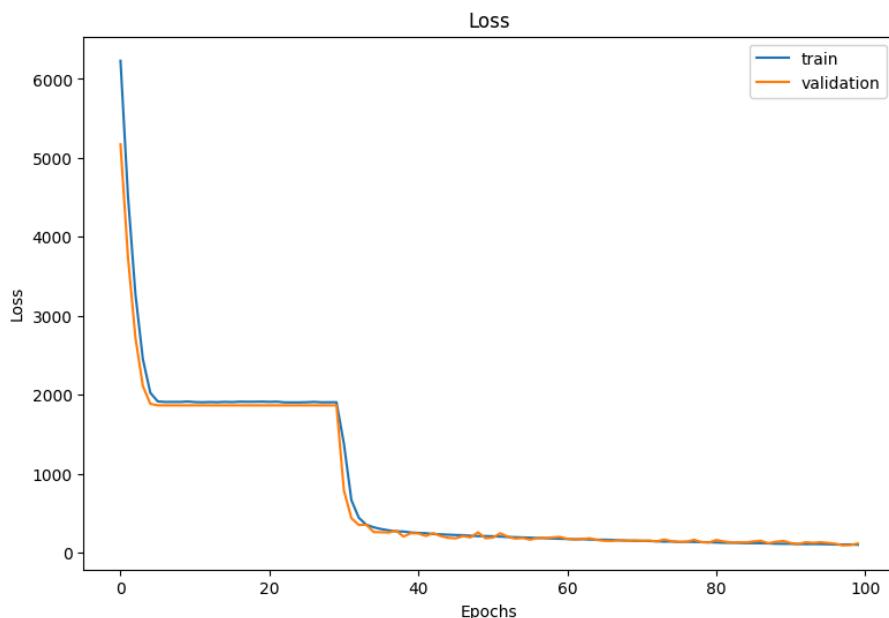
```

نتایج این بخش که 100 ایپاک آموزش دید به شکل زیر بود: (توجه کنید که با توجه به اینکه کل 100 تا آموزش دیده شد مدل شروع به overfitting نکرده که دلیل آن همانگونه که بالاتر توضیح داده شد به خاطر استفاده از بقیه روش‌هاست، در حالتی که از dropout و استفاده نشود مدل آموزش داده شد و بسیار overfitting رخ داد برای همین در حالت پایه نیز از dropout و استفاده شده تفاوت استفاده از early stopping حداقل در 100 ایپاک قابل مشاهده نباشد)

```
Epoch 100/100
222/222 [=====] - 4s 18ms/step - loss:
101.8150 - mean_squared_error: 99.4134 - val_loss: 118.1681 -
val_mean_squared_error: 115.7708
```

نتایج

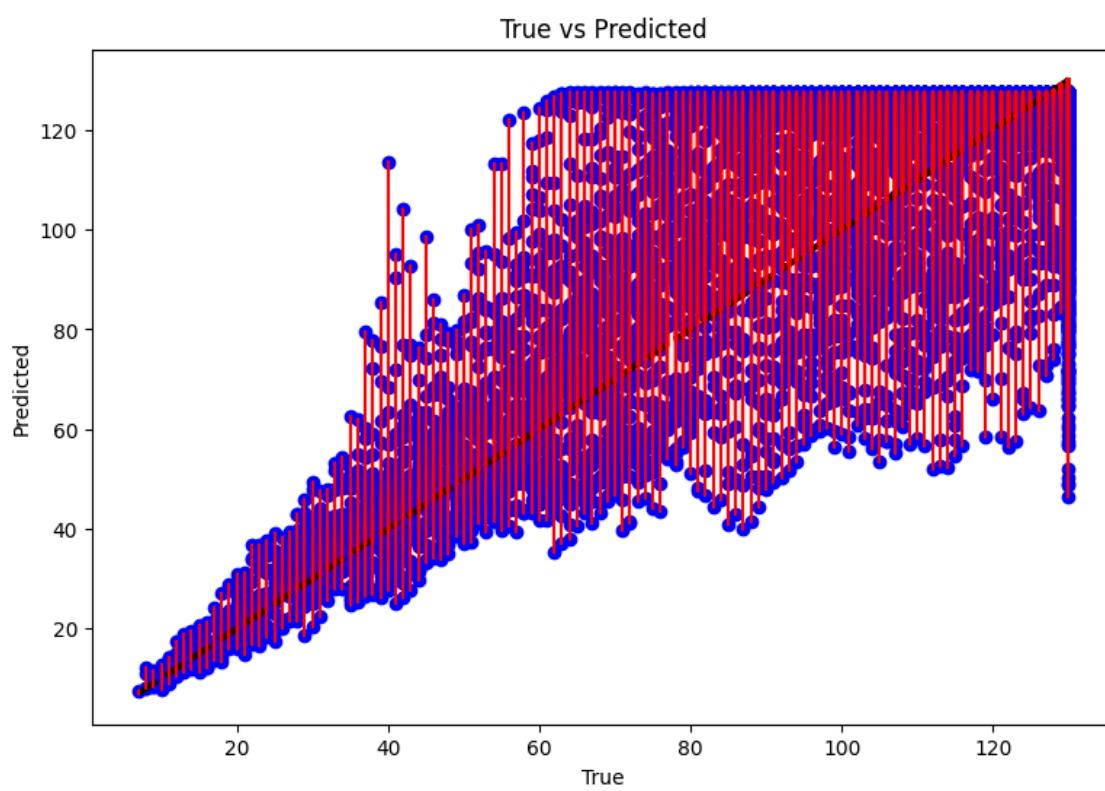
حال به بررسی نتایج این مدل می‌پردازیم:



شکل 2-14. نمودار آموزش برای مدل هایبرید regression پیشنهادی

در حالت استفاده از تمام n -window

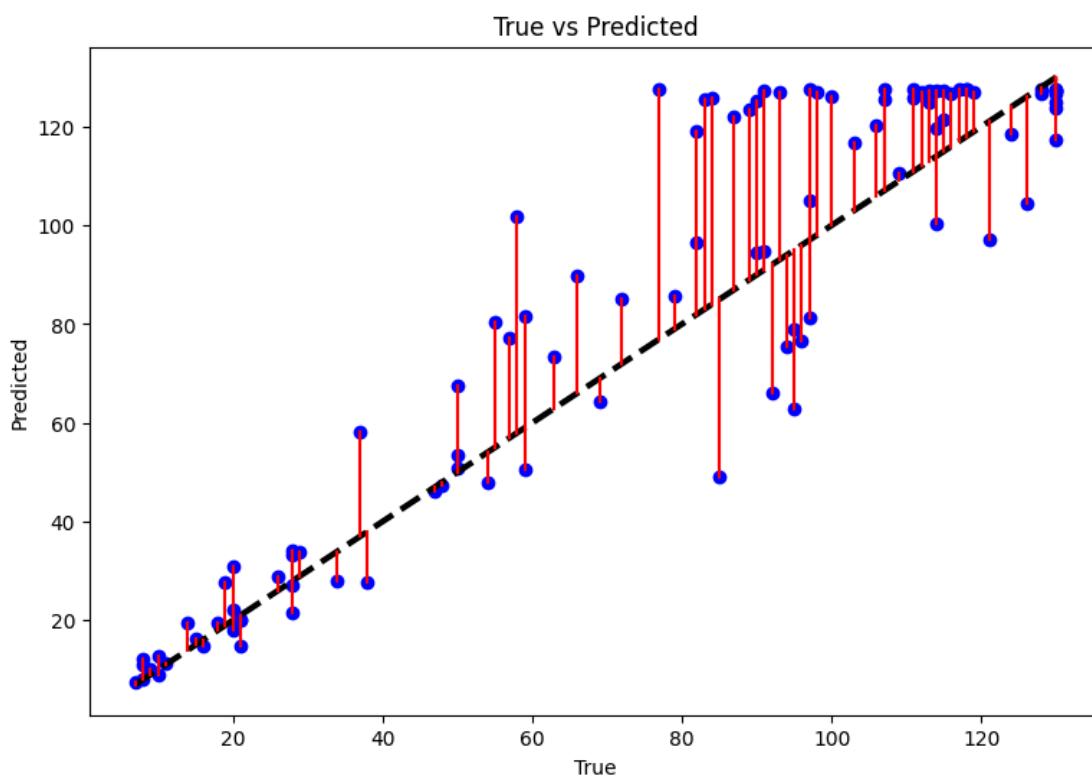
```
Mean Squared Error for Hybrid CNN LSTM Regression Model With  
Early Stopping: 308.8393252911665  
Mean Absolute Error for Hybrid CNN LSTM Regression Model With  
Early Stopping: 11.679724052028124  
Root Mean Squared Error for Hybrid CNN LSTM Regression Model  
With Early Stopping: 17.57382500456763  
Mean Absolute Percentage Error for Hybrid CNN LSTM Regression  
Model With Early Stopping: 0.13339969912801755
```



شکل 2-15. نمودار RUL-های واقعی و تخمین زده شده در حالت استفاده از تمام n -window
ها برای مدل هایبریدی پیشنهادی

در حالت استفاده تنها از آخرین window-ها

```
Mean Squared Error for Hybrid CNN LSTM Regression Model With Early Stopping (Last Window): 310.3870923218385  
Mean Absolute Error for Hybrid CNN LSTM Regression Model With Early Stopping (Last Window): 12.841083707809448  
Root Mean Squared Error for Hybrid CNN LSTM Regression Model With Early Stopping (Last Window): 17.617806115457125  
Mean Absolute Percentage Error for Hybrid CNN LSTM Regression Model With Early Stopping (Last Window): 0.19160851197033832
```



شکل 2-16. نمودار RUL-های واقعی و تخمین زده شده در حالت استفاده از آخرین window برای مدل هایبریدی پیشنهادی

از نتایج بدست آمده و نمودارها می‌توان نتیجه گرفت که مدل به خوبی در حال آموزش است و دقت مدل بسیار بالاست. هر چند که این نکته قابل توجه است که هر چقدر بیشتری مانده باشد مدل با دقت کمتری می‌تواند حدس بزنید که این با منطق پیش‌بینی آینده دور که سخت‌تر از آینده نزدیک است همخوانی دارد. می‌توان به عنوان راه حل از راه حل مقاوم استفاده کرد و خروجی مدل regression را به مدل classification داد و دقت را بالا برد.

3-2. مقایسه با مدل‌های پایه

حال می‌خواهیم بررسی کنیم چقدر مدل پیشنهادی خوب عمل می‌کند و آن را با چند تا از مدل‌های پایه دیگر مقایسه کنیم.

LSTM

Model: "sequential_19"

Layer (type)	Output Shape	Param #
<hr/>		
=		
lstm_41 (LSTM)	(None, 30, 64)	20992
dropout_43 (Dropout)	(None, 30, 64)	0
lstm_42 (LSTM)	(None, 30, 64)	33024
dropout_44 (Dropout)	(None, 30, 64)	0
lstm_43 (LSTM)	(None, 64)	33024
dropout_45 (Dropout)	(None, 64)	0
dense_21 (Dense)	(None, 1)	65
<hr/>		
=		
Total params: 87,105		
Trainable params: 87,105		
Non-trainable params: 0		

Classification

```
def initialize_LSTM_classification_model(  
    input_shape: tuple,  
    dropout_rate: float = 0.2,  
    l2_regularizer: float = 0.01
```

```

) -> models.Sequential:
    model = Sequential()
    model.add(
        layers.LSTM(
            64,
            input_shape=input_shape,
            kernel_regularizer=regularizers.l2(l2_regularizer),
            return_sequences=True,
        )
    )
    model.add(Dropout(dropout_rate))
    model.add(
        layers.LSTM(
            64,
            kernel_regularizer=regularizers.l2(l2_regularizer),
            return_sequences=False,
        )
    )
    model.add(Dropout(dropout_rate))
    model.add(Dense(1, activation="sigmoid"))

    model.compile(optimizer=Adam(learning_rate=0.001),
loss="binary_crossentropy", metrics=["accuracy"])
    return model

```

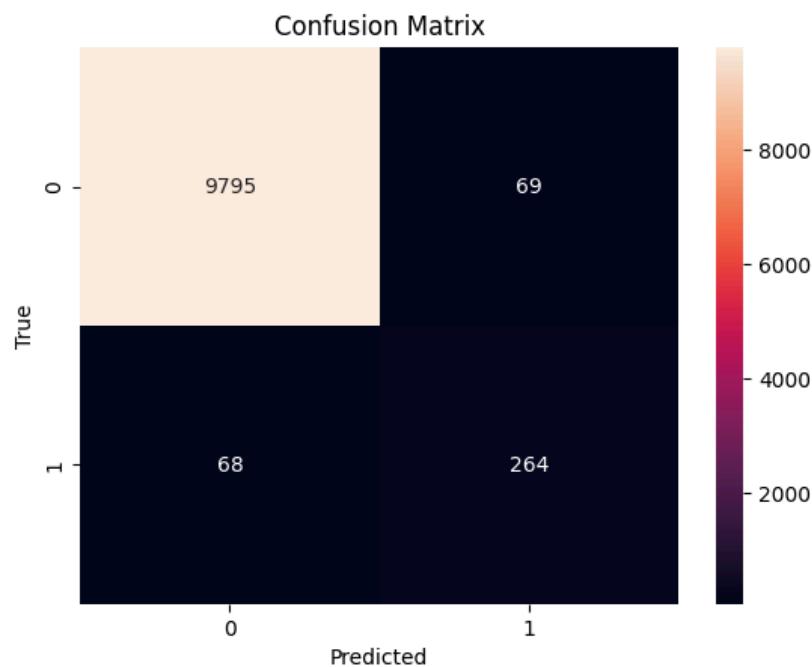
Epoch 19/100
222/222 [=====] - 2s 10ms/step - loss:
0.0854 - accuracy: 0.9688 - val_loss: 0.0846 - val_accuracy:
0.9676

	precision	recall	f1-score	support
--	-----------	--------	----------	---------

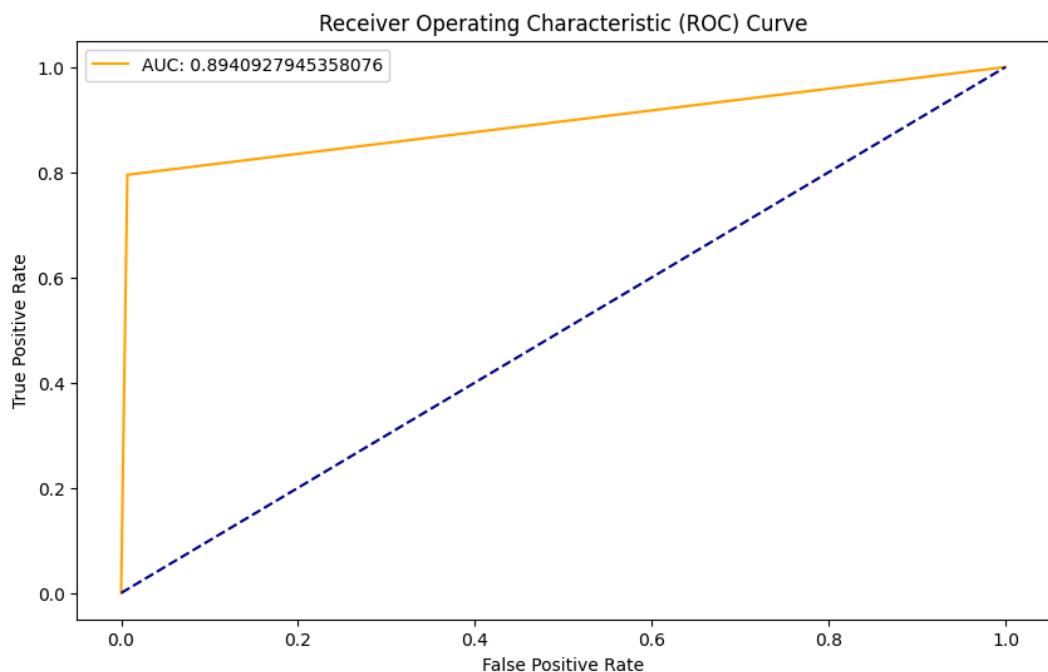
0.0	0.99	0.99	0.99	9864
1.0	0.79	0.80	0.79	332

accuracy	0.99	10196
----------	------	-------

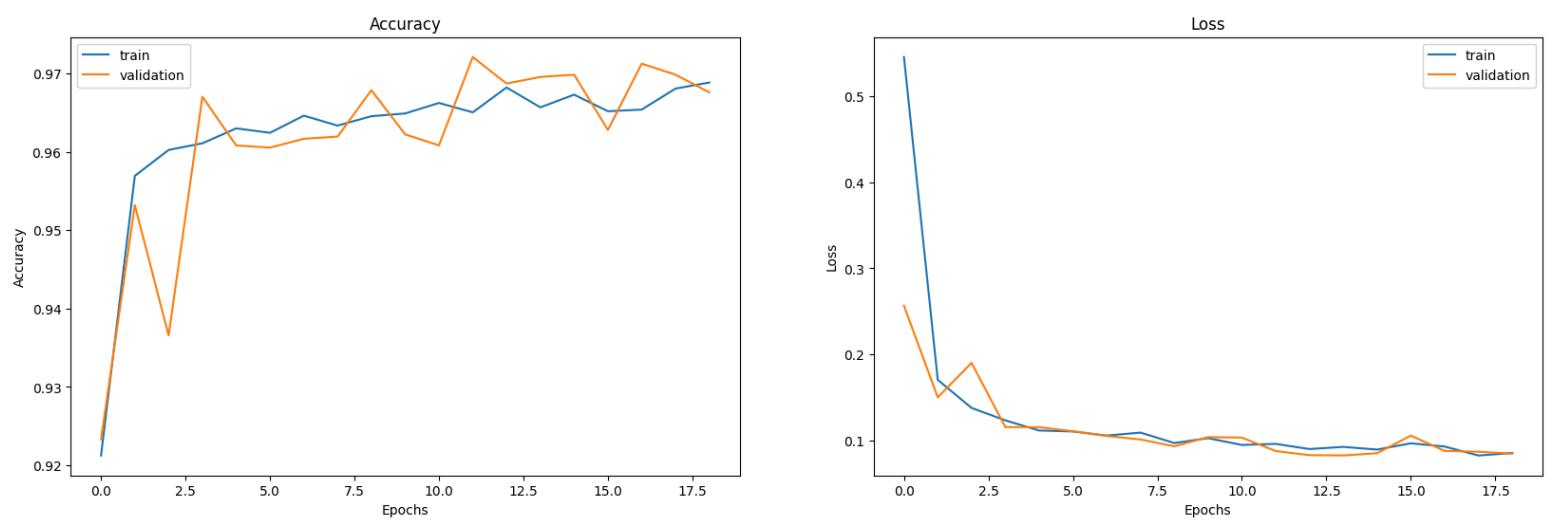
macro avg	0.89	0.89	0.89	10196
weighted avg	0.99	0.99	0.99	10196



شکل ۲. LSTM classification برای مدل Confusion Matrix .17-2



شکل 2-19. نمودار ROC برای مدل بدون LSTM classification



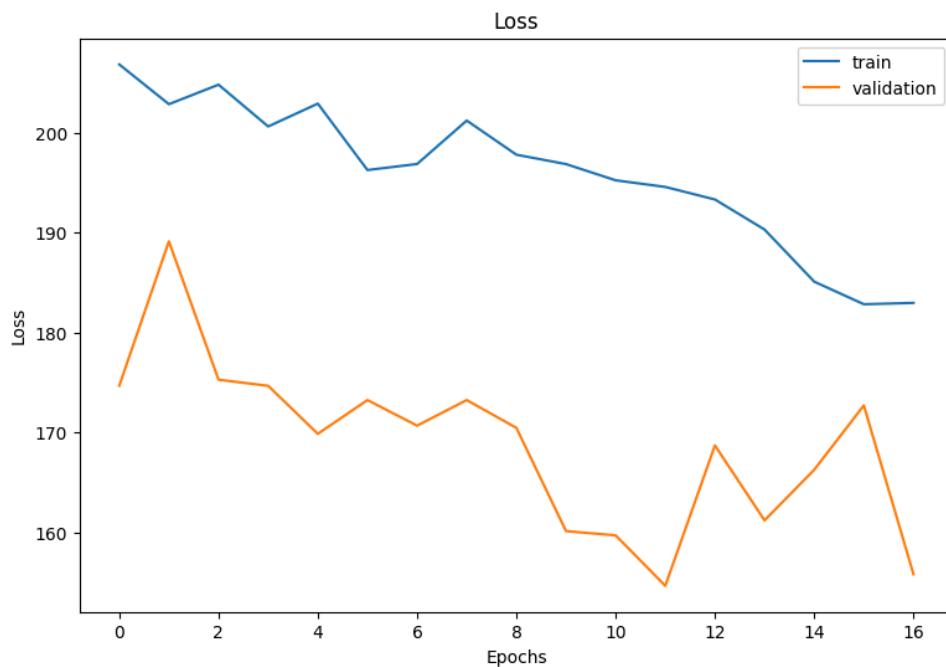
شکل 2-20. نمودار آموزش برای مدل LSTM regression

Regression

```
def initialize_LSTM_regression_model(
    input_shape: tuple,
    dropout_rate: float = 0.2,
    l2_regularizer: float = 0.01
) -> models.Sequential:
    model = Sequential()
    model.add(
        layers.LSTM(
            64,
            input_shape=input_shape,
            kernel_regularizer=regularizers.l2(l2_regularizer),
            return_sequences=True,
        )
    )
    model.add(Dropout(dropout_rate))
    model.add(
        layers.LSTM(
            64,
            kernel_regularizer=regularizers.l2(l2_regularizer),
            return_sequences=True,
        )
    )
    model.add(Dropout(dropout_rate))
    model.add(
        layers.LSTM(
            64,
            kernel_regularizer=regularizers.l2(l2_regularizer),
            return_sequences=False,
        )
    )
    model.add(Dropout(dropout_rate))
    model.add(Dense(1, activation="linear"))

    model.compile(optimizer=RMSprop(learning_rate=0.001),
    loss="mean_squared_error", metrics=["mean_squared_error"])
    return model
```

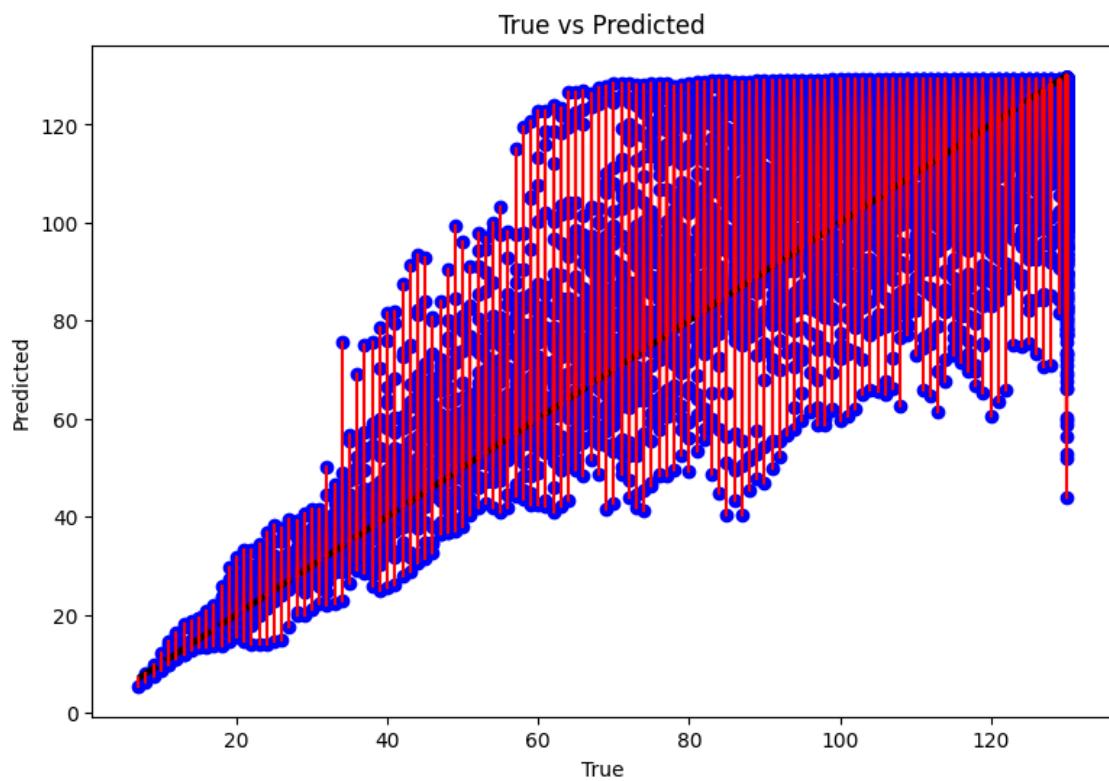
```
Epoch 17/100
222/222 [=====] - 3s 14ms/step - loss:
182.9688 - mean_squared_error: 179.7553 - val_loss: 155.8115 -
val_mean_squared_error: 152.5912
```



شکل 21-2. نمودار آموزش برای مدل LSTM regression

در حالت استفاده از تمام **LSTM**-window‌ها

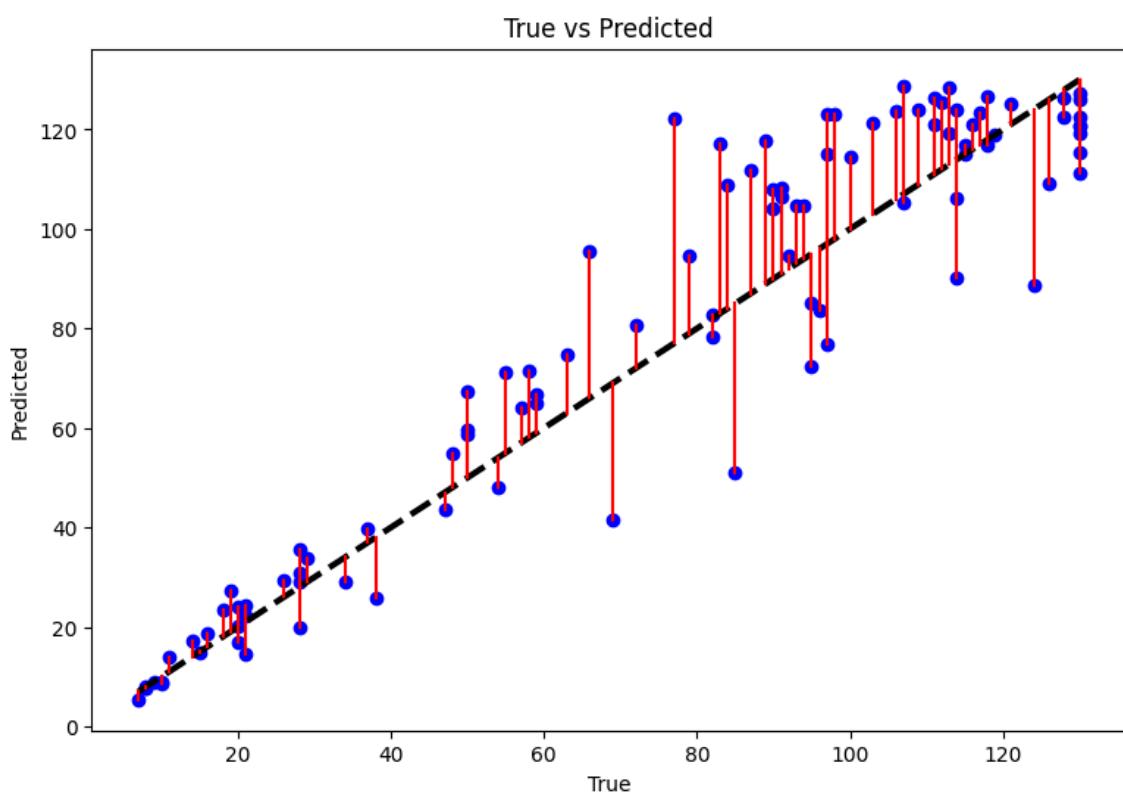
```
Mean Squared Error for LSTM Regression Model: 233.6239454669936
Mean Absolute Error for LSTM Regression Model:
10.729241820446132
Root Mean Squared Error for LSTM Regression Model:
15.284761871452025
Mean Absolute Percentage Error for LSTM Regression Model:
0.11911922930686888
```



شکل 2-22. نمودار RUL-های واقعی و تخمین زده شده در حالت استفاده از تمام LSTM-ها برای مدل window

در حالت استفاده تنها از آخرین window-ها

```
Mean Squared Error for LSTM Regression Model (Last Window):  
201.00986622097008  
Mean Absolute Error for LSTM Regression Model (Last Window):  
10.639924902915954  
Root Mean Squared Error for LSTM Regression Model (Last Window):  
14.177794829273347  
Mean Absolute Percentage Error for LSTM Regression Model (Last Window): 0.1546780800997162
```



شکل 2-23. نمودار RUL-های واقعی و تخمین زده شده در حالت استفاده از آخرین

LSTM برای مدل window

CNN

Model: "sequential_8"

Layer (type)	Output Shape	Param #
<hr/>		
=		
conv1d (Conv1D)	(None, 28, 64)	3328
max_pooling1d (MaxPooling1D)	(None, 14, 64)	0
conv1d_1 (Conv1D)	(None, 12, 64)	12352
max_pooling1d_1 (MaxPooling 1D)	(None, 6, 64)	0
conv1d_2 (Conv1D)	(None, 4, 64)	12352
max_pooling1d_2 (MaxPooling 1D)	(None, 2, 64)	0
flatten (Flatten)	(None, 128)	0
dense_8 (Dense)	(None, 64)	8256
dropout_24 (Dropout)	(None, 64)	0
dense_9 (Dense)	(None, 1)	65
<hr/>		
=		
Total params: 36,353		
Trainable params: 36,353		
Non-trainable params: 0		

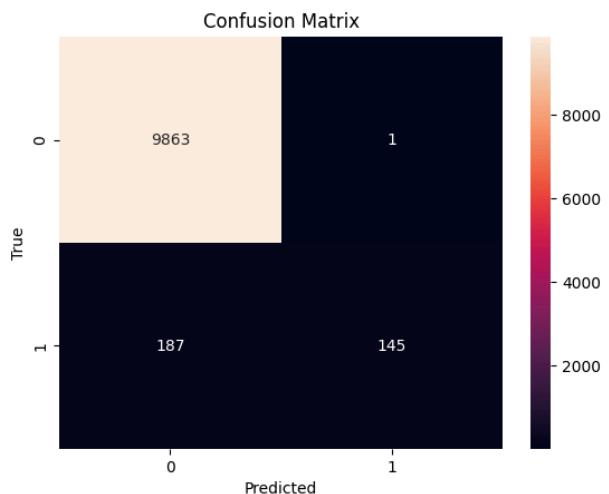
Classification

```
def initialize_CNN_classification_model(
    input_shape: tuple,
    dropout_rate: float = 0.2,
    l2_regularizer: float = 0.01
) -> models.Sequential:
    model = Sequential()
    model.add(layers.Conv1D(64, 3, activation="relu",
input_shape=input_shape))
    model.add(layers.MaxPooling1D(2))
    model.add(layers.Conv1D(64, 3, activation="relu"))
    model.add(layers.MaxPooling1D(2))
    model.add(layers.Conv1D(64, 3, activation="relu"))
    model.add(layers.MaxPooling1D(2))
    model.add(layers.Flatten())
    model.add(Dense(64, activation="relu",
kernel_regularizer=regularizers.l2(l2_regularizer)))
    model.add(Dropout(dropout_rate))
    model.add(Dense(1, activation="sigmoid"))

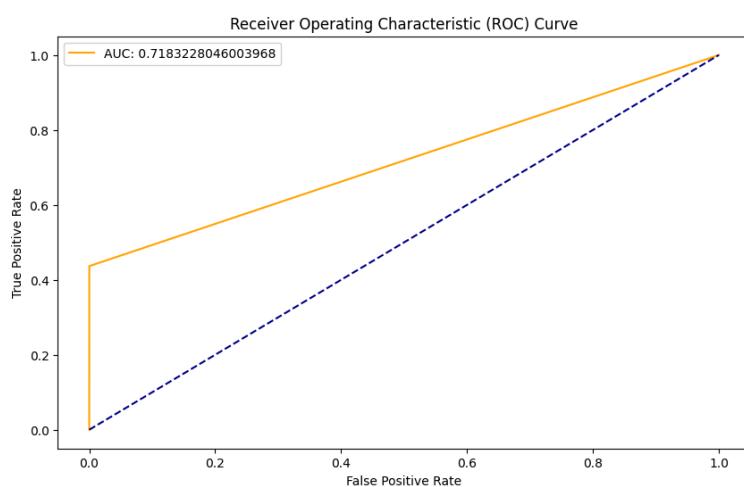
    model.compile(optimizer=Adam(learning_rate=0.001),
loss="binary_crossentropy", metrics=["accuracy"])
    return model
```

```
Epoch 22/100
222/222 [=====] - 1s 6ms/step - loss: 0.0469 - accuracy: 0.9841 - val_loss: 0.1291 - val_accuracy: 0.9529
```

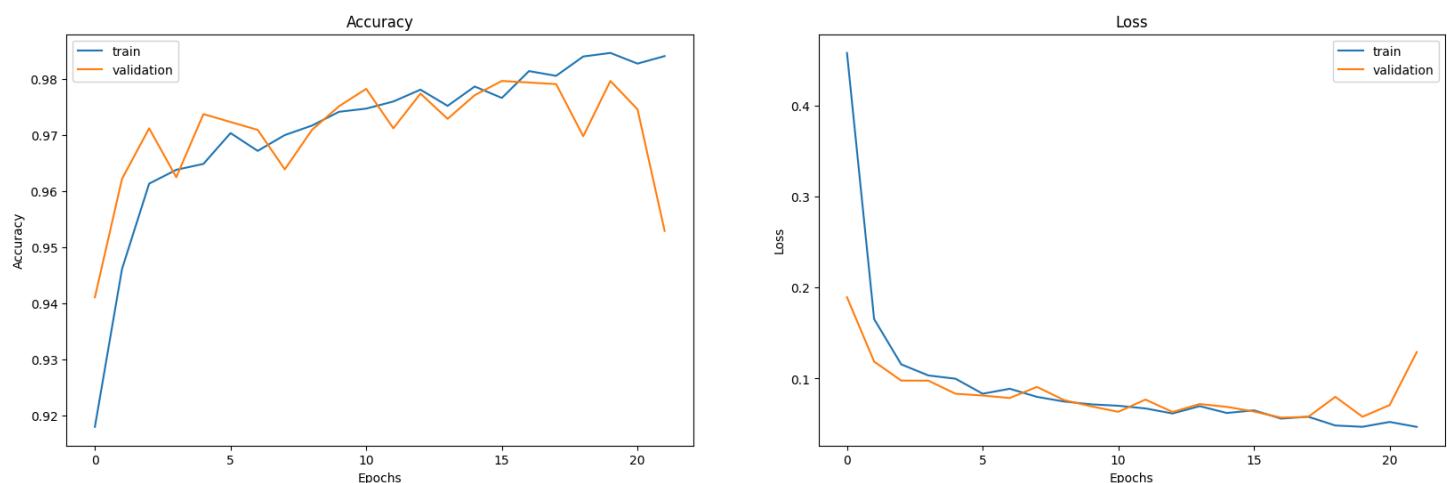
	precision	recall	f1-score	support
0.0	0.98	1.00	0.99	9864
1.0	0.99	0.44	0.61	332
accuracy			0.98	10196
macro avg	0.99	0.72	0.80	10196
weighted avg	0.98	0.98	0.98	10196



شکل 24-2. CNN classification برای مدل Confusion Matrix.



شکل 25-2. نمودار ROC برای مدل CNN classification



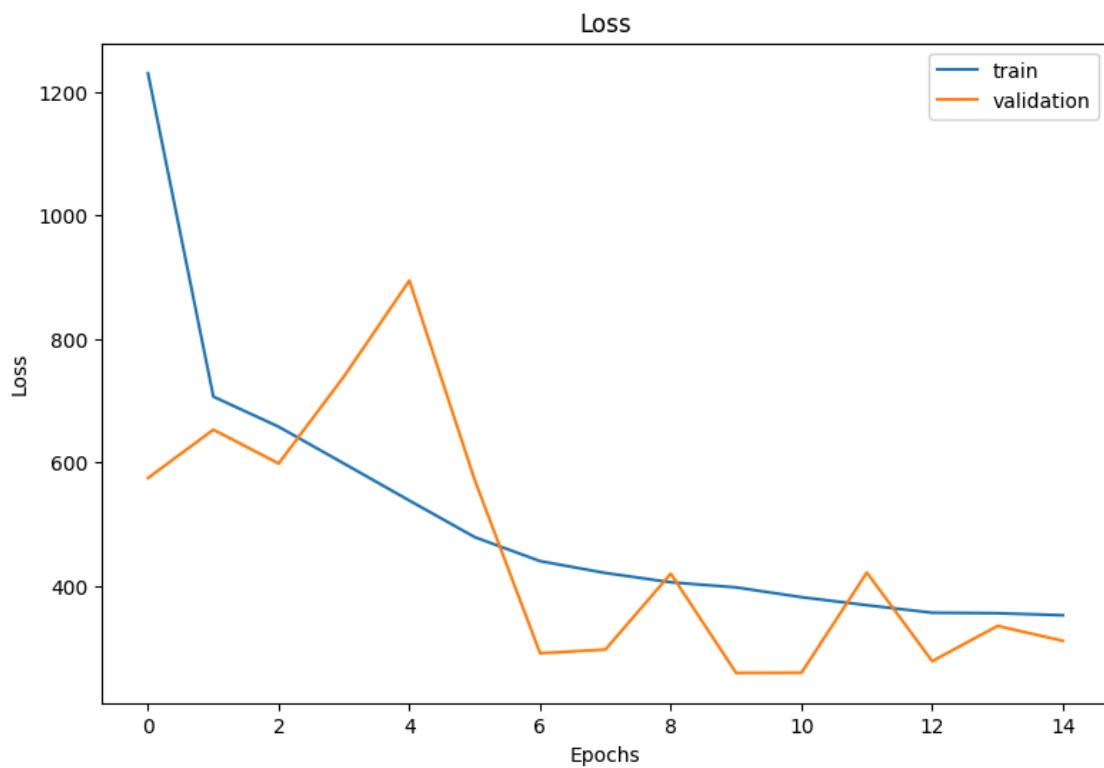
شکل 26-2. نمودار آموزش برای مدل CNN regression

Regression

```
def initialize_CNN_regression_model(
    input_shape: tuple,
    dropout_rate: float = 0.2,
    l2_regularizer: float = 0.01
) -> models.Sequential:
    model = Sequential()
    model.add(layers.Conv1D(64, 3, activation="relu",
input_shape=input_shape))
    model.add(layers.MaxPooling1D(2))
    model.add(layers.Conv1D(64, 3, activation="relu"))
    model.add(layers.MaxPooling1D(2))
    model.add(layers.Conv1D(64, 3, activation="relu"))
    model.add(layers.MaxPooling1D(2))
    model.add(layers.Flatten())
    model.add(Dense(64, activation="relu",
kernel_regularizer=regularizers.l2(l2_regularizer)))
    model.add(Dropout(dropout_rate))
    model.add(Dense(1, activation="linear"))

    model.compile(optimizer=RMSprop(learning_rate=0.001),
loss="mean_squared_error", metrics=["mean_squared_error"])
    return model
```

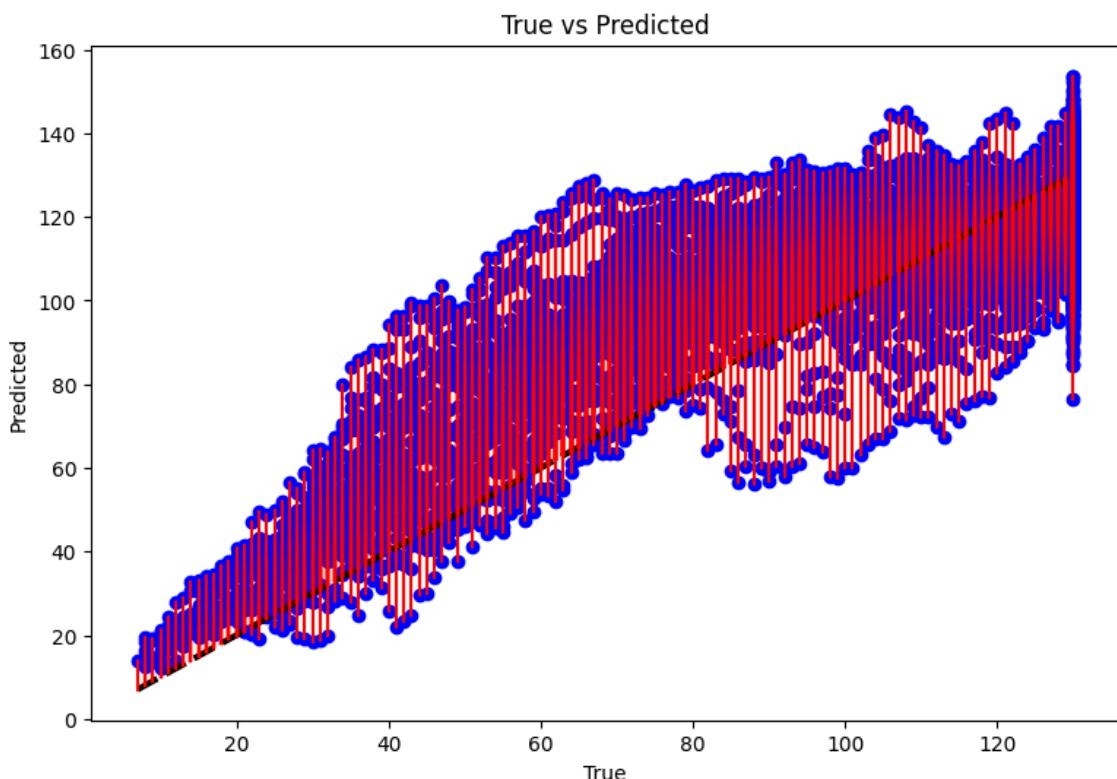
```
Epoch 15/100
222/222 [=====] - 1s 5ms/step - loss: 352.0591 - mean_squared_error: 351.5618 - val_loss: 310.5782 - val_mean_squared_error: 310.0811
```



شکل 2-27. نمودار آموزش برای مدل CNN regression

در حالت استفاده از تمام window-ها

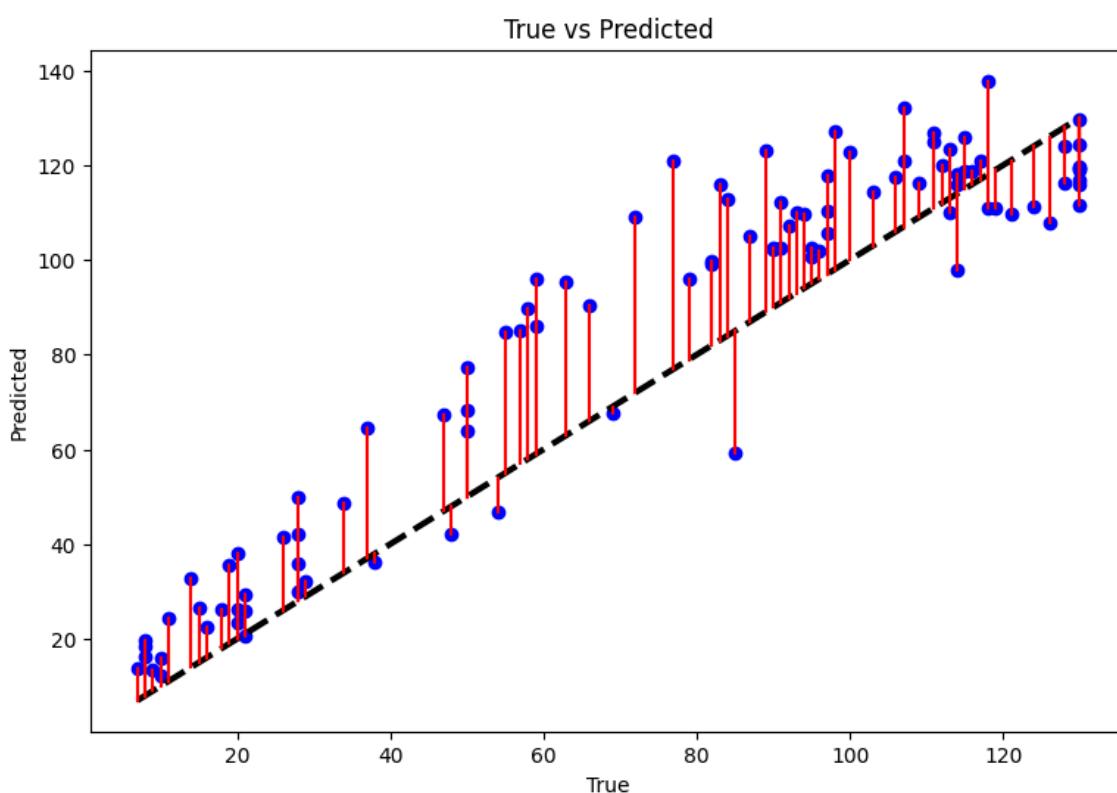
Mean Squared Error for CNN Regression Model: 282.47325233206027
Mean Absolute Error for CNN Regression Model: 13.054367483620926
Root Mean Squared Error for CNN Regression Model:
16.806940600003923
Mean Absolute Percentage Error for CNN Regression Model:
0.1656683892430355



شکل 2-28. نمودار RUL-های واقعی و تخمین زده شده در حالت استفاده از تمام CNN-ها برای مدل window

در حالت استفاده تنها از آخرین window-ها

```
Mean Squared Error for CNN Regression Model (Last Window):  
287.80194566109026  
Mean Absolute Error for CNN Regression Model (Last Window):  
14.11270993232727  
Root Mean Squared Error for CNN Regression Model (Last Window):  
16.964726513006045  
Mean Absolute Percentage Error for CNN Regression Model (Last Window): 0.30712176052202866
```



شکل 2-29. نمودار RUL-های واقعی و تخمین زده شده در حالت استفاده از آخرین CNN برای مدل window

مقایسه

همانگونه که دیده شد مدل‌های LSTM و CNN نیز دقیق‌تر قابل قبولی دارند اما مدل پیشنهادی در همه بخش‌ها بهتر عمل کرده. در جدول‌های زیر می‌توانیم نتایج را با هم مقایسه کنیم:

	Model	MSE	MAE	RMSE	MAPE
0	CNN Regression Model	282.473252	13.054367	16.806941	0.165668
1	LSTM Regression Model	295.883384	11.840559	17.201261	0.132063
2	Hybrid CNN LSTM Regression Model	233.623945	10.729242	15.284762	0.119119

شکل 2-30. مقایسه مدل‌های regression با تمام window-ها

	Model	MSE	MAE	RMSE	MAPE
0	CNN Regression Model	287.801946	14.112710	16.964727	0.307122
1	LSTM Regression Model	271.761140	11.638843	16.485179	0.169026
2	Hybrid CNN LSTM Regression Model	201.009866	10.639925	14.177795	0.154678

شکل 2-31. مقایسه مدل‌های regression با آخرین window

	Model	Confusion Matrix	Precision	Recall	AUC
0	CNN Regression Model	[[9863, 1], [187, 145]]	[0.032561788936837976, 0.9931506849315068, 1.0]	[1.0, 0.4367469879518072, 0.0]	0.718323
1	LSTM Regression Model	[[9795, 69], [68, 264]]	[0.032561788936837976, 0.7927927927927928, 1.0]	[1.0, 0.7951807228915663, 0.0]	0.894093
2	Hybrid CNN LSTM Regression Model	[[9794, 70], [61, 271]]	[0.032561788936837976, 0.7947214076246334, 1.0]	[1.0, 0.8162650602409639, 0.0]	0.904584

شکل 2-32. مقایسه مدل‌های classification با تمام window-ها

	Model	Confusion Matrix	Precision	Recall	AUC
0	CNN Regression Model	[[75, 0], [10, 15]]	[0.25, 1.0, 1.0]	[1.0, 0.6, 0.0]	0.800000
1	LSTM Regression Model	[[74, 1], [3, 22]]	[0.25, 0.9565217391304348, 1.0]	[1.0, 0.88, 0.0]	0.933333
2	Hybrid CNN LSTM Regression Model	[[73, 2], [2, 23]]	[0.25, 0.92, 1.0]	[1.0, 0.92, 0.0]	0.946667

شکل 2-33. مقایسه مدل‌های classification با آخرین window