

بسمه تعالیٰ



دانشگاه تهران

دانشکده مهندسی مکانیک

تمرین سری پنجم درس هوش مصنوعی

استاد:

جناب دکتر مسعود شریعت پناهی

نام و نام خانوادگی:

محمد Mehdi تویسرکانی

شماره دانشجویی:

۸۱۰۶۰۳۰۰۵

تاریخ تحویل:

۱۴۰۴/۰۴/۲۵

نیم‌سال دوم سال تحصیلی ۱۴۰۴-۱۴۰۳

## فهرست مطالب

صفحه	عنوان
۱	اطلاعات مسئله
۶	حل مسئله
۷	بخش الف: پیاده‌سازی مدل‌ها
۷	(۱) آماده‌سازی داده‌ها
۷	(۱-۱) بارگذاری و انتخاب دادگان
۷	(۱-۱-۱) بارگذاری زیرمجموعه FD001
۹	(۱-۱-۲) شناسایی و حذف ویژگی‌های با مقدار ثابت یا تغییرات بسیار کم
۱۰	(۱-۱-۳) شناسایی ویژگی‌های تکراری یا بسیار مشابه و نگه داشتن یکی از آنها
۱۱	(۱-۱-۴) بررسی ارتباط ویژگی‌ها با متغیر هدف و حذف ویژگی‌های با همبستگی بسیار پایین
۱۲	(۱-۱-۵) شناسایی و حذف ویژگی‌های با اهمیت پایین
۱۵	(۲-۱) تقسیم‌بندی داده‌ها با روش پنجره لغزان (Sliding Window)
۱۶	(۲-۳-۱) برچسب گذاری پنجره‌ها با مقدار RUL
۲۱	(۲-۴) تقسیم داده‌ها به آموزش و آزمون
۲۲	(۲-۵) نرمال‌سازی داده‌ها
۲۴	(۳) طراحی شبکه عصبی پیچشی (CNN)
۲۶	(۳) پیاده‌سازی شبکه LSTM
۲۸	(۴) تنظیم ابرپارامترها (Hyperparameter Tuning)
۲۸	(۴-۱) تنظیم ابرپارامترها برای شبکه CNN
۳۳	(۴-۲) تنظیم ابرپارامترها برای شبکه LSTM

۳۸	۵) پیاده‌سازی مدل‌های ترکیبی
۳۸	۱-۵) پیاده‌سازی مدل ترکیبی CNN + LSTM
۴۰	۲-۵) پیاده‌سازی مدل ترکیبی LSTM + CNN
۴۲	۶) پیاده‌سازی معماری LSTM + Attention
۴۴	بخش ب: ارزیابی مدل‌ها
۴۴	۱) نمودارها و مقایسه بصری
۵۴	۲) معیارهای ارزیابی مدل
۵۶	۳) تحلیل عملکرد (خلاصه)
۵۷	نتیجه‌گیری
۵۸	گیت هاب

## هوش مصنوعی و یادگیری ماشین (شبکه‌های بازگشتی)

### اطلاعات مسئله:

هدف این تمرین آشنایی با شبکه‌های بازگشتی و مقایسه عملکرد آن‌ها با شبکه‌های پیشرو در مدل‌سازی پدیده‌های وابسته به زمان است. برای این کار از مجموعه داده C-MAPSS استفاده خواهیم کرد که یکی از مجموعه داده‌های پرکاربرد ناسا در حوزه پایش وضعیت موتور توربوفن هواپیما است. این مجموعه داده با استفاده از یک شبیه‌ساز پیشرفته تولید شده است و شامل شرایط کاری موتور در طول چندین پرواز پیاپی و کیفیت عملکرد موتور در این شرایط است. در هر پرواز، حسگرها پارامترهای مختلفی مانند دما، فشار و شرایط کاری موتور را با نرخ نمونه‌برداری ۱ هرتز ثبت می‌کنند. همچنین در بخشی از پروازها، به صورت شبیه‌سازی شده خرابی‌هایی در اجزای مختلف موتور مانند کمپرسور، توربین و فن ایجاد می‌شود تا بتوان مدل‌ها را برای پیش‌بینی خرابی‌های احتمالی و زمان وقوع آن‌ها آموزش داد.

داده‌ها به صورت سری زمانی هستند و هر داده شامل ۳۰ پارامتر مختلف از موتور و شرایط پرواز است. زیرمجموعه FD001 ساده‌ترین حالت این دادگان است که تنها شامل یک نوع خرابی است. برای کاهش پیچیدگی مساله، در ادامه این تمرین از همین زیرمجموعه استفاده خواهیم کرد. توصیه می‌شود برای انجام تمرین از محیط Jupyter Notebook یا Google Colab استفاده کنید. مجموعه داده C-MAPSS و مقاله‌ای که آن را معرفی می‌کند در فایل ZIP تمرین قرار داده شده است.

### C-MAPSS (NASA Turbofan Engine)

#### بخش الف: پیاده‌سازی مدل‌ها

##### مرحله ۱: آماده‌سازی داده‌ها

###### بارگذاری و انتخاب دادگان

ابتدا زیرمجموعه FD001 را بارگذاری کنید. سپس ویژگی‌های حسگر مرتبط را انتخاب کنید و ویژگی‌هایی را که اطلاعات مفیدی ارائه نمی‌دهند حذف نمایید. برای این کار:

- ویژگی‌هایی با مقدار ثابت یا تغییرات بسیار کم (Low Variance Features) را حذف کنید (ویژگی‌هایی که تقریباً همیشه مقدار ثابتی دارند).
- ویژگی‌های تکراری یا بسیار مشابه (Duplicate or Highly Similar Features) را شناسایی و یکی از آن‌ها را نگه دارید.
- ارتباط ویژگی‌ها با متغیر هدف (Correlation with Target, Correlation Matrix) را بررسی کنید و ویژگی‌هایی با همبستگی بسیار پایین را کنار بگذارید.
- با استفاده از اهمیت ویژگی‌ها بر اساس مدل (Feature Importance, e.g., Random Forest Feature Importance) ویژگی‌هایی با اهمیت پایین را حذف نمایید. (امتیازی)

## تقسیم داده‌ها به آموزش و آزمون (Train/Test Split)

داده‌ها را به دو بخش ۷۰ درصدی (آموزش) و ۳۰ درصدی (آزمون) تقسیم کنید.

## نرمال‌سازی داده‌ها (Normalization)

در این مرحله، همه مقادیر عددی به یک بازه یا توزیع مشخص تبدیل می‌شوند تا مدل بهتر و سریع‌تر آموزش بییند. با استفاده از یکی از دو روش زیر، داده‌ها را نرمال‌سازی کنید:

- Min-Max: انتقال داده‌ها به بازه‌ای مشخص مثل ۰ تا ۱
- استانداردسازی به طوری که داده‌ها میانگین صفر و انحراف معیار یک داشته باشند Z-score

## تقسیم‌بندی داده‌ها با روش پنجره لغزان (Sliding Window)

روش «پنجره لغزان» برای تقسیم داده‌های سری زمانی به قسمت‌های کوچک‌تر و هماندازه به کار می‌رود. یک پنجره (مثلاً شامل ۳۰ چرخه پیاپی) را به تدریج روی داده‌ها حرکت دهید و در هر مرحله، داده‌های درون پنجره را استخراج کنید. این روش:

- داده‌ها را به نمونه‌هایی با طول یکسان تبدیل می‌کند که برای مدل‌های CNN و LSTM ضروری است.
- به مدل امکان می‌دهد الگوها و تغییرات کوتاه‌مدت در داده‌ها را بهتر تشخیص دهد.
- تعداد نمونه‌های قابل استفاده را افزایش می‌دهد و یادگیری مدل را بهبود می‌بخشد.

## برچسب‌گذاری پنجره‌ها با مقدار RUL

برای هر پنجره استخراج شده، یک برچسب "عمر باقی‌مانده RUL" تعیین کنید. برای این کار:

- برای هر لحظه (چرخه) در سری زمانی، حساب کنید که چند چرخه تا پایان عمر موتور باقی مانده است.
- برای هر پنجره، مقدار RUL مربوط به آخرین چرخه آن پنجره را انتخاب کرده و به کل پنجره نسبت دهید. به این ترتیب مدل یاد می‌گیرد که با مشاهده کل یک پنجره پیش‌بینی کند که در انتهای آن پنجره چقدر از عمر موتور باقی مانده است.
- اگر مقادیر RUL خیلی بزرگ باشد ممکن است آموزش مدل با چالش روپرورد شود. به همین دلیل می‌توانید یک حد بالا (مثلاً ۱۳۰ چرخه) برای RUL تعیین کنید و همه مقادیر بزرگ‌تر از آن را برابر با این حد قرار دهید. این کار به مدل کمک می‌کند روی پیش‌بینی‌های واقع‌بینانه‌تر تمرکز کند و دقت پیش‌بینی افزایش یابد.

## مرحله ۲: پیاده‌سازی شبکه عصبی پیچشی (CNN)

برای پیش‌بینی باقی‌مانده عمر مفید بر اساس داده‌های سری‌زمانی از یک CNN یک بعدی با معماری زیر استفاده کنید:

وروودی با شکل (اندازه پنجره، تعداد ویژگی‌ها):

Conv1D با ۶۴ فیلتر، کرنل ۳، فعال‌سازی ReLU، پدینگ

MaxPooling1D با اندازه پنجره ۲

Conv1D با ۱۲۸ فیلتر، کرنل ۳، فعال‌سازی ReLU، پدینگ

GlobalAveragePooling1D

ReLU با ۶۴ نورون، فعالسازی Dense

Dropout با نرخ ۰/۲

RUL با یک نورون برای خروجی Dense

آموزش:

تابع خطا: میانگین مربعات خطای (MSE) •

بهینه‌ساز: Adam با نرخ یادگیری ۰/۰۰۱ •

اندازه دسته (batch size) : ۳۲ یا ۶۴ •

تعداد دوره (epochs) : ۵۰ تا ۱۰۰ •

معیار ارزیابی: خطای میانگین قدر مطلق (MAE) و به صورت اختیاری log RMSE •

### مرحله ۳: پیاده‌سازی شبکه LSTM

برای پیش‌بینی باقی‌مانده عمر مفید بر اساس داده‌های سری زمانی از یک شبکه استاندارد LSTM با معماری زیر استفاده کنید:

ورودی با شکل (اندازه پنجره، تعداد ویژگی‌ها):

LSTM با ۱۰۰ واحد، بازگرداندن توالی

Dropout با نرخ ۰/۲

LSTM با ۵۰ واحد، فقط خروجی آخرین گام

ReLU با ۶۴ نورون، فعالسازی Dense

Dropout با نرخ ۰/۲

RUL با یک نورون برای مقدار Dense

کامپایل و آموزش:

تابع خطا: MSE •

بهینه‌ساز: Adam با نرخ یادگیری ۰/۰۰۱ •

اندازه دسته و تعداد دوره مشابه مدل CNN برای فراهم شدن امکان مقایسه •

### مرحله ۴: تنظیم ابرپارامترها (Hyperparameter Tuning)

پارامترهای زیر را با جستجوی دستی یا با استفاده از ابزارهایی مانند KerasTuner یا Optuna تنظیم کنید:

نرخ یادگیری •

اندازه دسته (batch size) •

تعداد فیلترها یا واحدها •

Dropout نرخ •

- اندازه پنجره
- نوع بهینه‌ساز (مقایسه Adam و RMSprop)

**گزارش:**

- کدام پارامترها بیشترین تأثیر را بر عملکرد مدل داشتند؟
- جدولی یا گزارشی از اجرای مدل با تنظیمات مختلف و نتایج ارائه دهید.
- درباره تعادل بین زمان آموزش، دقت و پیچیدگی مدل بحث کنید.

**مرحله ۵: پیاده‌سازی مدل ترکیبی CNN + LSTM**

این مدل ترکیبی با معماری زیر از توانایی CNN در استخراج ویژگی‌ها و توانایی LSTM در یادگیری زمانی بهره می‌برد.

ورودی با شکل (اندازه پنجره، تعداد ویژگی‌ها):

CNN با ۶۴ فیلتر، کرنل ۳، فعال‌سازی ReLU ، پدینگ مساوی  
MaxPooling1D با اندازه ۲  
LSTM با ۱۰۰ واحد، بدون بازگرداندن توالی  
Dropout با نرخ ۰/۳  
ReLU با ۶۴ نورون، فعال‌سازی Dense  
RUL با یک نورون برای خروجی Dense

این مدل را مشابه سایر مدل‌ها آموزش داده و ارزیابی کنید. می‌توانید ترتیب CNN و LSTM را نیز عوض کنید و نتیجه را بررسی نمایید. (امتیازی)

**مرحله ۶: پیاده‌سازی معماری LSTM + ATTENTION**

برای آشنایی با سازوکار توجه (Attention)، یک مدل شبکه عصبی LSTM با سازوکار توجه طراحی و پیاده‌سازی کنید. سپس این مدل را روی همان مجموعه داده آموزش دهید. شبکه LSTM قادر است وابستگی‌های زمانی و الگوهای بلندمدت را به خوبی یاد بگیرد، اما ممکن است به تنها بخش‌های سری زمانی را به طور یکسان مورد توجه قرار دهد. سازوکار توجه به مدل اجازه می‌دهد که روی بخش‌های مهم‌تر داده‌های سری زمانی تمرکز کند و وزن کمتری به بخش‌های کم‌همیت اختصاص دهد.

این معماری ویژگی‌های جالبی دارد، از جمله:

- افزایش دقت پیش‌بینی: به دلیل توجه هدفمند به بخش‌های حیاتی و تعیین کننده در سری زمانی.
- تفسیرپذیری بیشتر: به کمک وزن‌هایی که مکانیزم توجه تولید می‌کند، می‌توان مشخص کرد کدام قسمت از داده‌ها بیشترین تأثیر را در پیش‌بینی دارد.
- بهبود یادگیری وابستگی‌های بلندمدت و ظرفی که ممکن است در معماری‌های ساده‌تر به خوبی یاد گرفته نشوند.
- عملکرد بهتر در داده‌های پیچیده‌ای که بخش‌هایی با اهمیت متفاوت دارند و نیازمند توجه گزینشی هستند.

## بخش ب: ارزیابی عملکرد مدل‌ها

### ۱- نمودارها و مقایسه بصری

- منحنی‌های خطای آموزش و اعتبارسنجی مثلًا  $MSE$  را برای هر مدل رسم کنید.
- مقدار واقعی در برابر مقدار پیش‌بینی شده  $RUL$  را روی داده‌های تست برای مدل‌ها رسم کنید.

### ۲- معیارهای ارزیابی مدل

برای هر مدلی که آموزش داده‌اید معیارهای زیر را محاسبه کنید و نتایج را به صورت جدول ارائه دهید تا مقایسه مدل‌ها ساده‌تر انجام شود:

#### • ریشه میانگین مربعات خطای (RMSE):

ارزیابی عملکرد مدل با تأکید بر خطاهای بزرگ‌تر.

#### • میانگین قدر مطلق خطای (MAE):

نشان‌دهنده میانگین خطای مطلق پیش‌بینی‌ها است.

#### • ضریب تعیین ( $R^2$ ):

ارزیابی میزان تطابق بین مقادیر واقعی و پیش‌بینی شده (هرچه نزدیک‌تر به ۱ بهتر است).

#### • زمان آموزش مدل:

زمان صرف شده برای آموزش مدل (بر حسب ثانیه یا دقیقه).

معیارهای تکمیلی (اختیاری برای یادگیری عمیق‌تر دانشجویان):

#### • خطای درصد مطلق میانگین (MAPE):

معیاری جهت مقایسه خطای صورت درصدی که برای ارزیابی دقیق‌تر کاربرد دارد.

#### • منحنی خطای تجمعی (Cumulative Error Curve):

تحلیل نحوه توزیع خطاهای و بررسی این که چه تعداد پیش‌بینی‌ها خطای کمتری نسبت به یک حد معین دارند.

نتایج را به صورت جدولی برای مقایسه راحت‌تر ارائه دهید.

### ۳. تحلیل عملکرد (خلاصه)

به سوالات زیر پاسخ دهید:

- کدام مدل دقیق‌ترین پیش‌بینی را ارائه داد؟ به نظر شما دلیل عملکرد بهتر آن چه بود؟
- آیا هیچ یک از مدل‌ها نشانه‌هایی از بیش‌بازش (Overfitting) داشت؟ چگونه متوجه شدید؟
- درباره تعادل بین دقت (Accuracy) و زمان آموزش بحث کنید.

## ❖ حل مسئله

پیش‌بینی (Remaining Useful Life (RUL) یا عمر باقی‌مانده سیستم‌های مهندسی، یک مسئله مهم در نگهداری پیش‌بینانه است. هدف از این پروژه، مدل‌سازی و پیش‌بینی RUL موتورهای توربوفن با استفاده از داده‌های سری زمانی عملکرد آنهاست. داده‌ها از مجموعه داده CMAPSS تهیه شده که توسط NASA ارائه شده است. در این داده‌ها، رفتار هر موتور از شروع عملیات تا خرابی کامل ثبت شده و ویژگی‌های سنسورهای مختلف مانند فشار، دما و سرعت در هر چرخه زمانی فراهم است. چالش اصلی این است که با استفاده از داده‌های فعلی هر موتور، بتوان تخمین دقیقی از چرخه‌های باقیمانده آن ارائه داد. این موضوع می‌تواند منجر به کاهش هزینه‌های تعمیر، افزایش ایمنی و بهبود برنامه‌ریزی تعمیرات در صنایع گردد.

برای حل این مسئله، یک فرآیند گام‌به‌گام شامل آماده‌سازی داده‌ها، طراحی مدل‌های پیش‌بینی، آموزش شبکه‌ها، و تنظیم ابرپارامترها پیاده‌سازی شده است. در بخش اول، داده‌های سری زمانی پاک‌سازی و پردازش شده‌اند. سپس با استفاده از تکنیک Sliding Window به قطعات هماندازه تقسیم شده و برچسب RUL برای هر پنجره محاسبه شده است. پس از نرمال‌سازی ویژگی‌ها، دو مدل CNN و LSTM به صورت جداگانه طراحی و آموزش داده شده‌اند. همچنین با استفاده از ابزار KerasTuner، تنظیم ابرپارامترها برای بهبود عملکرد مدل‌ها انجام گرفته است. در نهایت، مدل‌ها بر اساس معیارهایی مانند MAE و RMSE ارزیابی شده‌اند و بهترین ترکیب از نظر دقت و کارایی استخراج شده است.

## بخش الف: پیاده‌سازی مدل‌ها

### ۱) آماده‌سازی داده‌ها

#### ۱-۱) بارگذاری و انتخاب دادگان

##### ۱-۱-۱) بارگذاری زیرمجموعه FD001

برای انجام این پروژه، زیرمجموعه FD001 از مجموعه داده CMAPSS استفاده شده است. این زیرمجموعه شامل داده‌های سری زمانی مربوط به عملکرد تعداد زیادی از موتورهای توربوفن در شرایط کاری یکنواخت و بار ثابت است. سه بخش اصلی دیتاست شامل موارد زیر است:

##### ۱-۱-۱-۱) Train Set (داده‌های آموزشی)

شکل داده‌ها: (20631,26) یعنی ۲۰۶۳۱ ردیف و ۲۶ ستون.

- Unit: شماره موتور (یا واحد). هر موتور یک شناسه یکتا دارد.

- Time: شماره چرخه (cycle) نشان‌دهنده زمان یا تعداد دفعات کارکرد.

- .(Operational Setting) تنظیمات عملیاتی op\_setting\_1 تا op\_setting\_3

- sensor\_1 تا sensor\_21: خروجی‌های ۲۱ حسگر مختلف روی موتور.

##### ۱-۱-۱-۲) Test Set (داده‌های آزمایشی)

شکل داده‌ها: (13096,26)

- هر ردیف نشان‌دهنده شرایط یک موتور در یک زمان مشخص است.

##### ۱-۱-۱-۳) RUL Set (مقادیر واقعی RUL در داده‌های تست)

شکل داده‌ها: (100,2)

- ستون اول: شماره موتور (unit).

- ستون دوم: مقدار واقعی RUL (تعداد چرخه باقی‌مانده تا خرابی)

داده‌ها در قالب فایل متنی ارائه شده‌اند و با استفاده از کتابخانه‌هایی نظیر Pandas به محیط Python وارد شده‌اند. هر ردیف نشان‌دهنده یک چرخه کاری از یک موتور است و شامل شناسه موتور، شماره چرخه و مقادیر ویژگی‌های سنسوری می‌باشد. از بین کل ویژگی‌ها، تنها سنسورهای با اطلاعات معنی‌دار نگه داشته شده‌اند.

```
import pandas as pd
```

```
# Define column names
```

```
column_names = ['unit', 'time'] + [f'op_setting_{i}' for i in range(1, 4)] + [f'sensor_{i}' for i in range(1, 22)]
```

```
# Load the datasets
```

```
train_file = 'G:/Artificial_Intelligence/Tamrin/Tamrin5/6. Turbofan Engine Degradation Simulation Data Set/CMAPSSData/train_FD001.txt'
```

```
test_file = 'G:/Artificial_Intelligence/Tamrin/Tamrin5/6. Turbofan Engine Degradation Simulation Data Set/CMAPSSData/test_FD001.txt'
```

```
rul_file = 'G:/Artificial_Intelligence/Tamrin/Tamrin5/6. Turbofan Engine Degradation Simulation Data Set/CMAPSSData/RUL_FD001.txt'
```

```
train_df = pd.read_csv(train_file, sep=' ', header=None)
```

```
test_df = pd.read_csv(test_file, sep=' ', header=None)
```

```
rul_df = pd.read_csv(rul_file, sep=' ', header=None)
```

```
# Drop empty columns at the end
```

```
train_df.drop(train_df.columns[[26, 27]], axis=1, inplace=True)
```

```
test_df.drop(test_df.columns[[26, 27]], axis=1, inplace=True)
```

```
# Assign column names
```

```
train_df.columns = test_df.columns = column_names
```

```
# Show dataset shapes
```

```
print("Train Set Shape:", train_df.shape)
```

```
print("Test Set Shape:", test_df.shape)
```

```
print("RUL Set Shape:", rul_df.shape)
```

```
# Show sample rows
```

```
print("\n Train Set:")
```

```
display(train_df.head())
```

```
print("\n Test Set:")
```

```
display(test_df.head())
```

```
print("\n RUL values:")
```

```
display(rul_df.head())
```

مجموعه داده‌های Train و Test به صورت زیر است:

Train Set Shape: (20631, 26)

Test Set Shape: (13096, 26)

RUL Set Shape: (100, 2)

### ۱-۱-۲) شناسایی و حذف ویژگی‌های با مقدار ثابت یا تغییرات بسیار کم

برخی ویژگی‌ها در طول زمان مقدار ثابتی داشته یا تغییرات آن‌ها بسیار ناچیز است. این ویژگی‌ها هیچ اطلاعات مفیدی به مدل یادگیری نمی‌دهند و تنها منجر به افزایش پیچیدگی و زمان پردازش می‌شوند. برای شناسایی این ویژگی‌ها، انحراف معیار (Standard Deviation) آن‌ها محاسبه شده و اگر کمتر از یک آستانه تعیین شده باشد (1e-5)، حذف شده‌اند.

```
from sklearn.feature_selection import VarianceThreshold
```

```
# Select sensor features only (excluding unit and time)
```

```
sensor_features = train_df.iloc[:, 2:]
```

```
# Apply variance threshold
```

```
selector = VarianceThreshold(threshold=1e-5)
```

```
selector.fit(sensor_features)
```

```
# Get low-variance features
```

```
low_variance_features = sensor_features.columns[~selector.get_support()]
```

```
print("Removed Low Variance Features:", list(low_variance_features))
```

```
# Drop them from both train and test
```

```
train_df.drop(columns=low_variance_features, inplace=True)
```

```
test_df.drop(columns=low_variance_features, inplace=True)
```

Feature های حذف شده که دارای با مقدار ثابت یا تغییرات بسیار کم (انحراف معیار کمتر از 1e-5) هستند،

به صورت زیر به دست می‌آیند:

```
Removed Low Variance Features: ['op_setting_1', 'op_setting_2', 'op_setting_3', 'sensor_1',  
'sensor_5', 'sensor_6', 'sensor_10', 'sensor_16', 'sensor_18', 'sensor_19']
```

### ۱-۱-۳) شناسایی ویژگی‌های تکراری یا بسیار مشابه و نگه داشتن یکی از آنها

برخی از ویژگی‌های موجود ممکن است به‌شدت با یکدیگر همبسته باشند و اطلاعات مشابهی ارائه دهند. وجود چنین ویژگی‌هایی باعث افزونگی داده‌ها و افزایش احتمال بیش‌بازش مدل می‌شود. بنابراین، با محاسبه ماتریس همبستگی و حذف ویژگی‌هایی با ضریب همبستگی بسیار بالا (بالاتر از 0.95)، تنها یکی از آن‌ها نگه داشته شده و سایر موارد حذف شده‌اند.

```
import numpy as np

# Function to identify duplicate features
def find_duplicate_features(df):
    duplicates = []
    # Calculate the correlation matrix
    corr_matrix = df.corr().abs()
    # Get the upper triangle of the correlation matrix
    upper = corr_matrix.where(np.triu(np.ones(corr_matrix.shape), k=1).astype(bool))
    # Find features with high correlation (>0.95)
    for column in upper.columns:
        if any(upper[column] > 0.95):
            duplicates.append(column)
    return duplicates

# Find duplicate features in the train and test datasets separately
train_duplicates = find_duplicate_features(train_df)
test_duplicates = find_duplicate_features(test_df)

# Display the duplicate features
print("Duplicate or highly similar features in the training data:")
for feature in train_duplicates:
    print(feature)

print("\nDuplicate or highly similar features in the test data:")
for feature in test_duplicates:
    print(feature)

# Drop duplicate features from the original datasets
train_cleaned = train_df.drop(columns=train_duplicates, errors='ignore')
test_cleaned = test_df.drop(columns=test_duplicates, errors='ignore')

# Ensure the number of features in train and test datasets are equal
if train_cleaned.shape[1] > test_cleaned.shape[1]:
```

```

train_cleaned = train_cleaned.iloc[:, :test_cleaned.shape[1]]
elif test_cleaned.shape[1] > train_cleaned.shape[1]:
    test_cleaned = test_cleaned.iloc[:, :train_cleaned.shape[1]]

# Display the cleaned datasets
print("\nShape of training data after cleaning:", train_cleaned.shape)
print("Shape of test data after cleaning:", test_cleaned.shape)

```

های حذف شده در هر یک از مجموعه داده‌های Train و Test که دارای ویژگی‌های تکراری یا بسیار

مشابه (ضریب همبستگی بالاتر از 0.95) هستند، به صورت زیر به دست می‌آیند:

Duplicate or highly similar features in the training data:  
sensor\_14

Duplicate or highly similar features in the test data:

Shape of training data after cleaning: (20631, 15)

Shape of test data after cleaning: (13096, 15)

#### ۱-۴-۱) بررسی ارتباط ویژگی‌ها با متغیر هدف و حذف ویژگی‌های با همبستگی بسیار پایین

برای بررسی تأثیر هر ویژگی بر پیش‌بینی RUL، همبستگی بین آن ویژگی و مقدار RUL محاسبه شده است.

ویژگی‌هایی که همبستگی بسیار کمی با RUL دارند، در یادگیری مدل نقش چندانی ندارند و ممکن است نویز ایجاد کنند. این ویژگی‌ها با استفاده از آستانه همبستگی (کمتر از 0.05) حذف شده‌اند تا فضای ویژگی تمیزتر و مدل ساده‌تر شود. مزایای حذف ویژگی‌های با همبستگی پایین شامل کاهش نویز (noise)، جلوگیری از پیچیدگی مدل، تسريع آموزش مدل و بهبود دقت و عملکرد مدل نهایی است.

```

import numpy as np

# Compute correlation matrix
corr_matrix = train_df.iloc[:, 2: ].corr().abs()

# Take upper triangle of the correlation matrix
upper_triangle = corr_matrix.where(np.triu(np.ones(corr_matrix.shape), k=1).astype(bool))

# Identify features with correlation < 0.05
low_corr_features = [column for column in upper_triangle.columns if
any(upper_triangle[column] < 0.05)]

```

```
# Drop them from both train and test
print("Removed Low Correlated Features:", low_corr_features)
train_df.drop(columns=low_corr_features, inplace=True)
test_df.drop(columns=low_corr_features, inplace=True)
```

Feature های حذف شده که دارای همبستگی بسیار پایین (کمتر از 0.05) هستند، به صورت زیر به دست می‌آیند:

Removed Low Correlated Features: ['sensor\_9', 'sensor\_13']

### ۱-۱-۵) شناسایی و حذف ویژگی‌های با اهمیت پایین

با استفاده از روش Random Forest Feature Importance، اهمیت نسبی ویژگی‌ها نسبت به متغیر هدف سنجیده شده است. سپس ویژگی‌هایی که نقش ناچیزی در پیش‌بینی داشتند (دارای اهمیت کمتر از 0.01)، حذف شده‌اند. این فرآیند کمک می‌کند تا مدل ساده‌تر و قابل تعمیم‌تر باشد و از بیش‌برازش جلوگیری شود.

```
from sklearn.ensemble import RandomForestRegressor
```

```
# Create temporary RUL column: RUL = max(time) - current time
train_df['RUL'] = train_df.groupby('unit')['time'].transform(max) - train_df['time']
```

```
# Features for importance (excluding unit, time, RUL)
X_temp = train_df.drop(columns=['unit', 'time', 'RUL'])
y_temp = train_df['RUL']
```

```
# Fit Random Forest Regressor
rf = RandomForestRegressor(n_estimators=100, random_state=42)
rf.fit(X_temp, y_temp)
```

```
# Feature importances
importances = rf.feature_importances_
feature_importance_series = pd.Series(importances,
index=X_temp.columns).sort_values(ascending=False)
```

```
# Display all features ranked by importance
print("All Features Ranked by Importance:")
print(feature_importance_series)
```

```
# Optionally: remove features with very low importance (< 0.01)
low_importance_features = feature_importance_series[feature_importance_series <
0.01].index.tolist()
```

```
print("\n Features Removed due to Low Importance (<0.01):")
print(low_importance_features)
```

```
# Drop them
train_df.drop(columns=low_importance_features, inplace=True)
test_df.drop(columns=low_importance_features, inplace=True)
```

های حذف شده که دارای اهمیت پایین در پیش‌بینی متغیر هدف هستند (دارای اهمیت کمتر از 0.01)، به صورت زیر به دست می‌آیند:

All Features Ranked by Importance:

```
sensor_11    0.450903
sensor_14    0.125161
sensor_4     0.102381
sensor_12    0.052343
sensor_7     0.046419
sensor_15    0.042694
sensor_21    0.040541
sensor_3     0.037168
sensor_2     0.035001
sensor_20    0.030088
sensor_8     0.024941
sensor_17    0.012360
dtype: float64
```

Features Removed due to Low Importance (<0.01):

[]

جدول مرتب‌شده اهمیت ویژگی‌ها (Feature Importance) بر اساس خروجی مدل در ادامه ارائه شده است:

ردیف	ویژگی	Feature Importance
1	sensor_11	0.450903
2	sensor_14	0.125161
3	sensor_4	0.102381
4	sensor_12	0.052343
5	sensor_7	0.046419
6	sensor_15	0.042694
7	sensor_21	0.040541
8	sensor_3	0.037168
9	sensor_2	0.035001
10	sensor_20	0.030088

0.024941	sensor_8	11
0.012360	sensor_17	12

تمام ویژگی‌ها (features) با استفاده از معیار اهمیت براساس تأثیرشان در پیش‌بینی RUL مرتب شده‌اند. آستانه (threshold) برای اهمیت ویژگی‌ها برابر با 0.01 در نظر گرفته شده است. یعنی، اگر اهمیت یک ویژگی کمتر از 0.01 باشد، بی‌اهمیت تلقی می‌شود و حذف می‌گردد. در غیر این صورت، نگه داشته می‌شود. همانطور که از خروجی مشخص است، هیچ ویژگی‌ای اهمیت کمتر از 0.01 نداشته است. بنابراین، هیچ ویژگی‌ای در این مرحله حذف نشده است.

در ادامه، Feature‌های باقی مانده به غیر از unit و time در هر یک از مجموعه داده‌های Train و Test به دست می‌آیند:

```
# Get feature columns (excluding 'unit' and 'time')
train_features = set(train_df.columns) - {'unit', 'time'}
test_features = set(test_df.columns) - {'unit', 'time'}

# Find common features
common_features = sorted(list(train_features & test_features))

print("Common features kept in both train and test (", len(common_features), "):")
print(common_features)

# Keep only common features + unit + time
train_df = train_df[['unit', 'time']] + common_features
test_df = test_df[['unit', 'time']] + common_features
```

تعداد Feature‌های باقی مانده ۱۲ تاست که به صورت زیر هستند:

Common features kept in both train and test (12):

['sensor\_11', 'sensor\_12', 'sensor\_14', 'sensor\_15', 'sensor\_17', 'sensor\_2', 'sensor\_20', 'sensor\_21', 'sensor\_3', 'sensor\_4', 'sensor\_7', 'sensor\_8']

در نهایت، کل Feature‌های نهایی در هر یک از مجموعه داده‌های Train و Test بعد از حذفیات بخش‌های

۱-۱-۱ تا ۱-۱-۵ به دست می‌آیند:

```
print("Final Train Shape:", train_df.shape)
print("Final Test Shape:", test_df.shape)
print("Final Selected Features:", train_df.columns.tolist())
```

تعداد کل Feature های نهایی (به همراه unite و time) در هر یک از مجموعه داده های Train و Test بعد از

حذفیات بخش های ۱-۱-۲ تا ۱-۱-۵، ۱۴ تاست که به صورت زیر هستند:

Final Train Shape: (20631, 14)

Final Test Shape: (13096, 14)

Final Selected Features: ['unit', 'time', 'sensor\_11', 'sensor\_12', 'sensor\_14', 'sensor\_15',  
'sensor\_17', 'sensor\_2', 'sensor\_20', 'sensor\_21', 'sensor\_3', 'sensor\_4', 'sensor\_7', 'sensor\_8']

## (۲-۱) تقسیم‌بندی داده‌ها با روش پنجره لغزان (Sliding Window)

روش پنجره لغزان به منظور استخراج قطعات زمانی هماندازه از سری داده‌ها استفاده شده است. یک پنجره با اندازه ثابت ۳۰ چرخه روی داده‌های هر موتور حرکت می‌کند و در هر گام، داده‌های درون پنجره استخراج و به عنوان یک نمونه آموزشی ذخیره می‌شود. این کار باعث می‌شود مدل بتواند از الگوهای محلی استفاده کند. هم‌چنان، این روش به افزایش تعداد نمونه‌ها کمک کرده و مدل را در تشخیص تغییرات محلی داده‌ها توانمندتر می‌سازد. این روش برای مدل‌های CNN و LSTM ضروری است، چون آن‌ها به ورودی‌هایی با اندازه یکسان نیاز دارند.

```
import numpy as np

def generate_sliding_windows(df, window_size=30, max_rul=130):
    X = []
    y = []

    # Get feature columns
    feature_cols = df.columns.difference(['unit', 'time'])

    # Process per unit
    for unit_id in df['unit'].unique():
        unit_df = df[df['unit'] == unit_id].sort_values('time')
        unit_data = unit_df[feature_cols].values
        num_cycles = unit_data.shape[0]

        # Compute RUL for each cycle
        rul_values = np.flip(np.arange(num_cycles)) # reverse counting
```

```
rul_values = np.clip(rul_values, a_min=None, a_max=max_rul)

# Apply sliding window
for i in range(num_cycles - window_size + 1):
    window = unit_data[i:i + window_size]
    label = rul_values[i + window_size - 1]
    X.append(window)
    y.append(label)

return np.array(X), np.array(y)
```

### ۱-۳) برچسب گذاری پنجره‌ها با مقدار RUL

برای هر پنجره استخراج شده از سری زمانی، مقدار RUL (عمر باقی‌مانده) به عنوان برچسب تعیین شده است.

این مقدار برابر با تعداد چرخه‌های باقی‌مانده تا خرابی موتور از انتهای آن پنجره می‌باشد. با این رویکرد، مدل می‌آموزد که با مشاهده یک پنجره از داده‌ها، تخمین بزند چند چرخه دیگر موتور کار خواهد کرد. برای جلوگیری از پراکندگی زیاد در مقادیر برچسب، یک حد بالا (۱۳۰ چرخه) برای RUL در نظر گرفته شده و مقادیر بالاتر به آن مقدار محدود شده‌اند. این اقدام باعث پایداری بیشتر آموزش مدل می‌شود.

```
# Set window size and max RUL
WINDOW_SIZE = 30
MAX_RUL = 130

# Generate training samples
X_train, y_train = generate_sliding_windows(train_df, window_size=WINDOW_SIZE,
max_rul=MAX_RUL)
print(" Training set shape:", X_train.shape, y_train.shape)

# For test data, use the LAST window of each engine
def generate_last_test_windows(test_df, rul_df, window_size=30, max_rul=130):
    X_test = []
    y_test = []
    feature_cols = test_df.columns.difference(['unit', 'time'])

    for idx, unit_id in enumerate(test_df['unit'].unique()):
        unit_df = test_df[test_df['unit'] == unit_id].sort_values('time')
        unit_data = unit_df[feature_cols].values

        if len(unit_data) >= window_size:
```

```

last_window = unit_data[-window_size:]
else:
    padding = np.repeat(unit_data[0:1], window_size - len(unit_data), axis=0)
    last_window = np.concatenate([padding, unit_data], axis=0)

# Ensure we're accessing within bounds
rul_index = min(idx, len(rul_df) - 1)
rul = min(rul_df.iloc[rul_index, 0], max_rul)
X_test.append(last_window)
y_test.append(rul)

return np.array(X_test), np.array(y_test)

# Apply to test set
X_test, y_test = generate_last_test_windows(test_df, rul_df, window_size=WINDOW_SIZE,
max_rul=MAX_RUL)
print(" Test set shape:", X_test.shape, y_test.shape)

```

در این مرحله داده‌های حسگرهای سری زمانی را به پنجره‌هایی با طول ثابت (sliding windows) تقسیم کرده‌ایم و برای هر پنجره یک برچسب RUL مشخص کرده‌ایم. برای هر واحد (engine) در دیتاست، داده‌های حسگر را به صورت پنجره‌های متوالی به طول ثابت (در اینجا ۳۰ تایم استپ) برش زده‌ایم. یعنی به جای استفاده از کل دنباله، بخش‌هایی از آن با طول ۳۰ به مدل داده می‌شود. برای هر پنجره، مقدار RUL محاسبه شده است. این مقدار برابر است با تعداد سیکل‌های باقی‌مانده تا خرابی از انتهای آن پنجره. همچنین مقدار RUL را با محدود (clip) کرده‌ایم تا از تأثیر مقدارهای خیلی بزرگ جلوگیری شود.

تعداد Feature های باقی‌مانده در هر یک از مجموعه داده‌های Train و Test، بعد از تقسیم داده‌ها با روش پنجره لغزان (Sliding Window) و برچسب گذاری با مقدار RUL به صورت زیر به دست می‌آیند:

Training set shape: (17731, 30, 12)  
Test set shape: (100, 30, 12)

► داده‌های آموزش (Training Set)

: X\_train شکل آن (17321,30,12) است:

✓ 17731: تعداد کل پنجره‌های آموزشی (sliding windows).

✓ 30: طول هر پنجره (تعداد تایم استپ‌ها).

✓ 12: تعداد ویژگی‌های انتخاب شده (sensor features).

: شکل آن (17731) است:

✓ برای هر پنجره، یک عدد RUL به عنوان برچسب (label) وجود دارد.

#### » داده‌های تست (Test Set)

: شکل آن (100,30,12) است:

✓ فقط از آخرین پنجره برای هر یک از 100 واحد در دیتای تست استفاده شده. بنابراین 100 نمونه داریم،

هر کدام با 30 تایم‌استپ و 12 ویژگی.

: شکل آن (100,) است:

✓ مقدار واقعی RUL برای هر واحد در تست.

در ادامه، با توجه به اینکه برچسب گذاری داده‌ها با مقدار RUL انجام شده است، می‌توان محاسبه ماتریس همبستگی و رسم نمودار Heatmap آن را با استفاده از کد زیر انجام داد:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Convert sliding window data into DataFrame (average per window)
X_flat = X_train.mean(axis=1)
X_flat_df = pd.DataFrame(X_flat, columns=train_df.columns.difference(['unit', 'time']))

# Add RUL as target
y_series = pd.Series(y_train, name='RUL')

# Combine features and target
corr_df = pd.concat([X_flat_df, y_series], axis=1)

# Compute correlation matrix
corr_matrix = corr_df.corr()
```

```
# Extract and sort correlation with RUL
correlations_with_target = corr_matrix['RUL'].drop('RUL').sort_values(ascending=False)
print("Feature Correlation with Target (RUL):")
print(correlations_with_target)

# Plot heatmap
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(10, 8), dpi=300)
ax = sns.heatmap(corr_matrix, annot=True, fmt=".2f", cmap='coolwarm', square=True,
                  cbar_kws={"shrink": 1.0, "aspect": 20},
                  linewidths=0.5, linecolor='gray')

plt.title("Correlation Matrix Heatmap (Features + RUL)", fontsize=12)
plt.xticks(rotation=45, ha='right', fontsize=10)
ax.set_yticklabels(ax.get_yticklabels(), rotation=0, fontsize=10)

plt.tight_layout()
plt.savefig(r'G:\Artificial_Intelligence\Tamrin\tamrin5\Results\Heatmap_Correlation_Matrix.png')
plt.show()
```

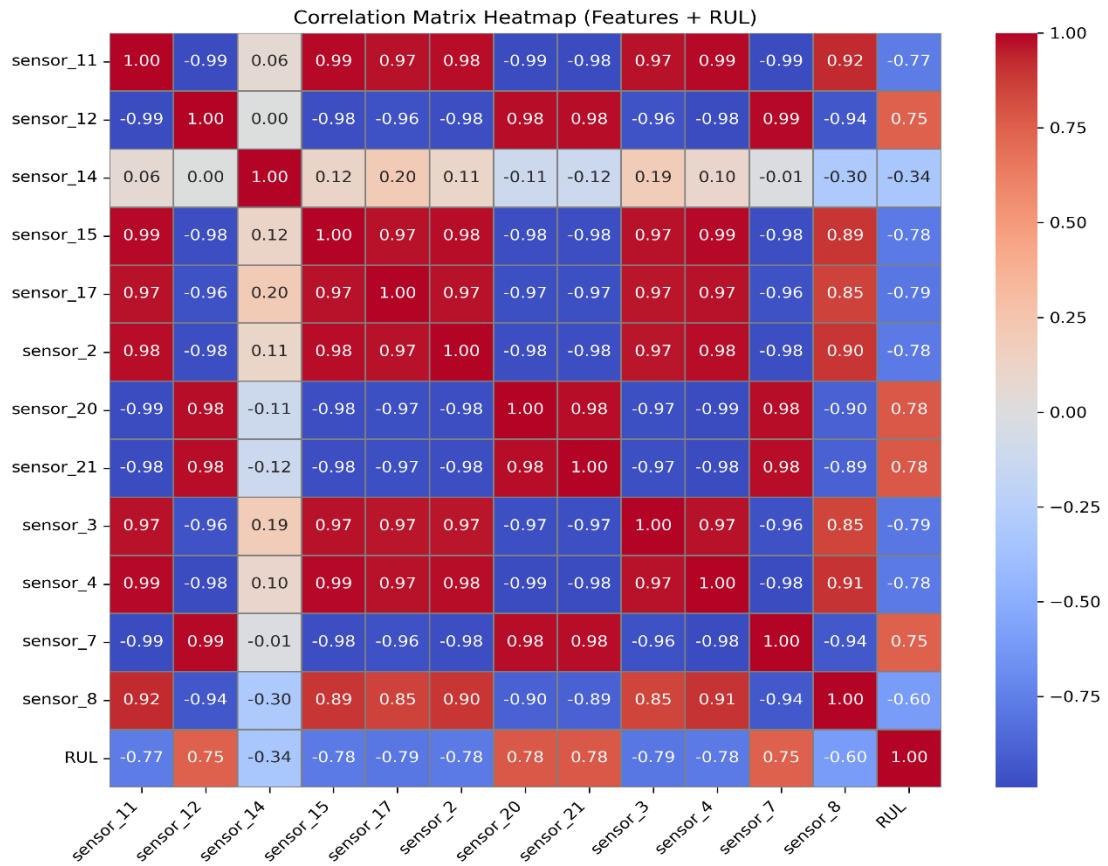
ماتریس همبستگی هر یک از Feature ها با متغیر هدف (RUL) و نمودار Heatmap آن به صورت زیر به دست

می‌آیند:

Feature Correlation with Target (RUL):

sensor_20	0.776686
sensor_21	0.776526
sensor_7	0.751067
sensor_12	0.746763
sensor_14	-0.335537
sensor_8	-0.602360
sensor_11	-0.767289
sensor_4	-0.775173
sensor_2	-0.775893
sensor_15	-0.779815
sensor_3	-0.789230
sensor_17	-0.793409

Name: RUL, dtype: float64



شکل ۱: نمودار Heatmap ماتریس همبستگی هر یک از Feature ها با مقدار RUL

جدول مرتب شده‌ی ویژگی‌ها بر اساس ضریب همبستگی (Correlation) با هدف RUL ارائه شده است:

Correlation with RUL	Feature
+0.7767	sensor_20
+0.7765	sensor_21
+0.7511	sensor_7
+0.7468	sensor_12
-0.3355	sensor_14
-0.6024	sensor_8
-0.7673	sensor_11
-0.7752	sensor_4
-0.7759	sensor_2
-0.7798	sensor_15
-0.7892	sensor_3
-0.7934	sensor_17

مقدار مثبت نشان می‌دهد که با افزایش مقدار ویژگی، مقدار RUL نیز افزایش می‌یابد، در حالی که مقادیر منفی نشان‌دهنده رابطه معکوس با RUL هستند.

#### ۴-۱ تقسیم داده‌ها به آموزش و آزمون

پس از آماده‌سازی و برچسب‌گذاری داده‌ها، آن‌ها به دو بخش آموزش (Training) و آزمون (Test) تقسیم شده‌اند. در این تمرین، ۷۰ درصد از داده‌ها برای آموزش مدل استفاده می‌شود و ۳۰ درصد باقی‌مانده برای ارزیابی عملکرد مدل به کار می‌رود. این تقسیم به گونه‌ای انجام شده که موتورهایی که در آموزش استفاده شده‌اند با موتورهای بخش آزمون تفاوت داشته باشند تا مدل توانایی تعمیم داشته باشد و صرفاً به داده‌های خاص وابسته نباشد.

```
from sklearn.model_selection import train_test_split

# Split train set into train and validation
X_train_final, X_val, y_train_final, y_val = train_test_split(
    X_train, y_train, test_size=0.3, random_state=42
)

print("Final Training set shape:", X_train_final.shape)
print("Validation set shape:", X_val.shape)
print("Test set shape:", X_test.shape)
```

مجموعه داده‌های Train و Validation بعد از تقسیم‌بندی داده‌ها به صورت زیر به دست می‌آیند:

Final Training set shape: (12411, 30, 12)  
Validation set shape: (5320, 30, 12)  
Test set shape: (100, 30, 12)

در جدول زیر، توضیحات تکمیلی مربوط به هر یک از مجموعه داده‌ها داده شده است:

توضیح	شکل داده	مجموعه داده
شامل 12411 پنجره زمانی است که هر کدام شامل 30 گام زمانی (داده‌های متوالی) و 12 ویژگی (سنسور) هستند. از این مجموعه برای یادگیری مدل استفاده می‌شود.	(12411, 30, 12)	Training
شامل 5320 پنجره زمانی است که ساختاری مشابه با داده‌های آموزش دارد. از این داده‌ها برای تنظیم هایپرپارامترها و جلوگیری از overfitting استفاده می‌شود.	(5320, 30, 12)	Validation

شامل 100 نمونه برای هر موتور (unit) است که فقط آخرین پنج روزه زمانی قبل از وقوع خرابی را شامل می‌شود. از این داده‌ها برای ارزیابی نهایی مدل استفاده می‌شود.

(100, 30, 12)

Test

## ۱-۵) نرمال‌سازی داده‌ها

ویژگی‌های داده‌ها با استفاده از روش Min-Max Scaling در بازه  $[0,1]$  نرمال‌سازی شده‌اند. این روش با حفظ ساختار توزیع داده‌ها، از مقیاس‌دهی متفاوت ویژگی‌ها جلوگیری می‌کند و باعث می‌شود مدل بهتر و سریع‌تر یاد بگیرد که رابطه آن به صورت زیر است:

$$X' = \frac{X - X_{\min}}{X_{\max} - X_{\min}}$$

در مقایسه با روش Z-score (نرمال‌سازی بر اساس میانگین و انحراف معیار)، روش Min-Max برای داده‌های دارای دامنه مشخص و بدون توزیع نرمال، مناسب‌تر است و از تأثیر مقادیر پرت نیز جلوگیری می‌کند. در این تمرین، چون داده‌ها فاقد توزیع نرمال هستند، Min-Max گزینه بهتری است. بعلاوه، با توجه به اینکه هدف این تمرین استفاده از شبکه‌های بازگشتی ( مثل LSTM) برای مدل‌سازی داده‌های زمان‌دار (سری زمانی) است، روش Min-Max مناسب‌تر است. مجموع دلایل آن در جدول زیر خلاصه شده است:

توضیح	عامل
شبکه‌هایی مثل LSTM معمولاً وقتی ورودی‌ها در بازه‌ای محدود مثل $[0,1]$ قرار دارند، پایداری و سرعت یادگیری بهتری دارند. Min-Max دقیقاً این ویژگی را فراهم می‌کند.	سازگاری با شبکه‌های عصبی بازگشتی
داده‌های سنسورها و شرایط پروازی لزوماً توزیع نرمال ندارند. روش Min-Max برخلاف Z-score فرضی درباره نوع توزیع داده نمی‌کند، بنابراین در این شرایط مناسب‌تر است.	عدم فرض توزیع نرمال
سیاری از ویژگی‌ها مثل دما یا فشار دارای بازه فیزیکی مشخصی هستند. روش Min-Max با نگه داشتن مقادیر بین حداقل و حداکثر، با این نوع داده‌ها سازگارتر است.	تناسب با داده‌های سنسوری دارای حدود مشخص
در اکثر مقالات و پیاده‌سازی‌های علمی در زمینه پیش‌بینی RUL با داده CMAPSS (مانند مقاله Zheng et al. 2017) از Min-Max LSTM استفاده شده و نتایج بهتری گرفته‌اند.	تجربه موفق در مطالعات مشابه

Min-Max ساختار و شکل نسبی داده‌ها را بهتر حفظ می‌کند که برای مدل‌سازی روند آموزش در زمان بسیار مهم است.

حفظ شکل توالی‌ها برای یادگیری  
سری زمانی

کد مربوط به نرمال‌سازی داده‌ها در ادامه آورده شده است:

```
from sklearn.preprocessing import MinMaxScaler

# Initialize Min-Max scaler on training data to avoid data leakage
scaler = MinMaxScaler()

# Reshape 3D data (samples, window_size, features) to 2D (total_timesteps, features)
num_samples, window_size, num_features = X_train_final.shape
X_train_2d = X_train_final.reshape(-1, num_features)
X_val_2d = X_val.reshape(-1, num_features)
X_test_2d = X_test.reshape(-1, num_features)

# Fit scaler only on training data
scaler.fit(X_train_2d)

# Transform train, validation, and test data
X_train_norm = scaler.transform(X_train_2d).reshape(num_samples, window_size,
num_features)
X_val_norm = scaler.transform(X_val_2d).reshape(X_val.shape)
X_test_norm = scaler.transform(X_test_2d).reshape(X_test.shape)

print("Data normalized using Min-Max scaling to range [0, 1].")
print("Normalized data shapes:", X_train_norm.shape, X_val_norm.shape, X_test_norm.shape)
```

در نهایت، داده‌ها با روش Min-Max نرمال‌سازی شده و مجموعه داده‌های نرمال شده در هر یک از مجموعه داده‌های Train، Test و RUL به صورت زیر هستند:

```
Data normalized using Min-Max scaling to range [0, 1].
Normalized data shapes: (12411, 30, 12) (5320, 30, 12) (100, 30, 12)
```

## ۲) طراحی شبکه عصبی پیچشی (CNN)

مدل CNN طراحی شده از یک معماری ساده اما قدرتمند برای تحلیل داده های سری زمانی استفاده می کند. این مدل شامل دو لایه کانولوشن با تعداد فیلترهای ۶۴ و ۱۲۸ است که به ترتیب به شناسایی الگوهای محلی و پیچیده تر کمک می کنند. پس از هر لایه کانولوشن، یک عملیات global pooling یا pooling برای کاهش ابعاد و جلوگیری از بیش بارازش انجام می شود. سپس داده ها وارد لایه های Dense و Dropout می شوند و در نهایت یک نورون خروجی مقدار RUL را پیش بینی می کند. از تابع فعال سازی ReLU و بهینه ساز Adam با نرخ یادگیری ۰.۰۰۱ استفاده شده است. مدل با تابع خطای MSE آموزش داده می شود و MAE به عنوان معیار ارزیابی اصلی ثبت شده است.

### » طراحی مدل CNN

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv1D, MaxPooling1D, GlobalAveragePooling1D, Dense, Dropout
from tensorflow.keras.optimizers import Adam

# Define input shape
input_shape = (X_train_norm.shape[1], X_train_norm.shape[2])

# Build the CNN model
model_cnn = Sequential([
    Conv1D(filters=64, kernel_size=3, activation='relu', padding='same',
           input_shape=input_shape),
    MaxPooling1D(pool_size=2),
    Conv1D(filters=128, kernel_size=3, activation='relu', padding='same'),
    GlobalAveragePooling1D(),
    Dense(64, activation='relu'),
    Dropout(0.2),
    Dense(1) # Output layer for RUL prediction
])

# Compile the CNN model
model_cnn.compile(
    loss='mse',
    optimizer=Adam(learning_rate=0.001),
    metrics=['mae']
```

)

model\_cnn.summary()

نتیجه‌ی طراحی مدل CNN را به صورت زیر می‌توان نمایش داد:

**Model: "sequential"**

Layer (type)	Output Shape	Param #
conv1d (Conv1D)	(None, 30, 64)	2,368
max_pooling1d (MaxPooling1D)	(None, 15, 64)	0
conv1d_1 (Conv1D)	(None, 15, 128)	24,704
global_average_pooling1d (GlobalAveragePooling1D)	(None, 128)	0
dense (Dense)	(None, 64)	8,256
dropout (Dropout)	(None, 64)	0
dense_1 (Dense)	(None, 1)	65

Total params: 35,393 (138.25 KB)

Trainable params: 35,393 (138.25 KB)

Non-trainable params: 0 (0.00 B)

## » آموزش مدل CNN

```
# Train the CNN model
import time

start_time = time.time()

history_cnn = model_cnn.fit(
    X_train_norm, y_train_final,
    validation_data=(X_val_norm, y_val),
    epochs=80,
    batch_size=32,
    verbose=1
)

# Calculate Training Time for CNN model
cnn_train_time = time.time() - start_time
print(f"Training time: {cnn_train_time:.2f} seconds")
```

زمان آموزش مدل نیز محاسبه شده است که 376.76 ثانیه به دست آمد.

Training time: 376.76 seconds

### ۳) پیاده‌سازی شبکه LSTM

شبکه LSTM طراحی شده شامل دو لایه متوالی LSTM با ۱۰۰ و ۵۰ واحد است که برای استخراج وابستگی‌های زمانی طولانی مدت در داده‌ها کاربرد دارد. لایه اول خروجی به صورت دنباله‌ای برمی‌گرداند تا لایه دوم بتواند وابستگی‌ها را بهتر درک کند. پس از آن لایه‌های Dense و Dropout به منظور افزایش توان یادگیری و کاهش بیش‌بازش اضافه شده‌اند. در انتهای، یک نورون خروجی برای پیش‌بینی مقدار RUL استفاده شده است. این مدل با همان تنظیمات بهینه‌ساز، تابع خطأ و تعداد دوره‌های مشابه CNN آموزش داده شده تا مقایسه‌ای منصفانه بین دو مدل انجام شود.

#### » طراحی مدل LSTM

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
from tensorflow.keras.optimizers import Adam

# Define Input shape
input_shape = (X_train_norm.shape[1], X_train_norm.shape[2])

# Build the LSTM model
model_lstm = Sequential([
    LSTM(100, return_sequences=True, input_shape=input_shape),
    Dropout(0.2),
    LSTM(50, return_sequences=False),
    Dense(64, activation='relu'),
    Dropout(0.2),
    Dense(1) # Output RUL
])
# Compile the LSTM model
model_lstm.compile(
    loss='mse',
    optimizer=Adam(learning_rate=0.001),
    metrics=['mae']
)
model_lstm.summary()
```

نتیجه‌ی طراحی مدل LSTM را به صورت زیر می‌توان نمایش داد:

Model: "sequential\_1"

Layer (type)	Output Shape	Param #
lstm (LSTM)	(None, 30, 100)	45,200
dropout_1 (Dropout)	(None, 30, 100)	0
lstm_1 (LSTM)	(None, 50)	30,200
dense_2 (Dense)	(None, 64)	3,264
dropout_2 (Dropout)	(None, 64)	0
dense_3 (Dense)	(None, 1)	65

Total params: 78,729 (307.54 KB)

Trainable params: 78,729 (307.54 KB)

Non-trainable params: 0 (0.00 B)

## » آموزش مدل LSTM

```
# Train the LSTM model
import time

start_time = time.time()

history_lstm = model_lstm.fit(
    X_train_norm, y_train_final,
    validation_data=(X_val_norm, y_val),
    epochs=80,
    batch_size=32,
    verbose=1
)

# Calculate Training Time for LSTM model
lstm_train_time = time.time() - start_time
print(f"Training time: {lstm_train_time:.2f} seconds")
```

زمان آموزش مدل نیز محاسبه شده است که 1525.29 ثانیه به دست آمد.

Training time: 1525.29 seconds

## (۴) تنظیم ابرپارامترها (Hyperparameter Tuning)

## ۴-۱) تنظیم ابرپارامترها برای شبکه CNN

برای بهبود عملکرد مدل CNN، با استفاده از ابزار KerasTuner تنظیماتی روی پارامترهایی مانند تعداد فیلترها، نرخ Dropout، نرخ Dense، نوع بهینه‌ساز انجام شده است تا بهترین پارامترها به منظور به حداقل رساندن خطای MAE روی داده‌های اعتبارسنجی (val\_mae) به دست آید. مشخص شد که اندازه پنجره و تعداد فیلترها تأثیر چشمگیری بر دقت مدل دارند. افزایش تعداد فیلترها معمولاً باعث بهبود عملکرد شد، ولی زمان آموزش را نیز افزایش داد. بهینه‌ساز Adam در اکثر موارد عملکرد بهتری نسبت به RMSprop نشان داد.

```
import keras_tuner as kt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv1D, MaxPooling1D, GlobalAveragePooling1D, Dense, Dropout
from tensorflow.keras.optimizers import Adam, RMSprop

def build_cnn_model(hp):
    model = Sequential()
    model.add(Conv1D(
        filters=hp.Choice('filters1', [64, 128]),
        kernel_size=3,
        activation='relu',
        padding='same',
        input_shape=(X_train_norm.shape[1], X_train_norm.shape[2]))
    )
    model.add(MaxPooling1D(pool_size=2))
    model.add(Conv1D(
        filters=hp.Choice('filters2', [64, 128]),
        kernel_size=3,
        activation='relu',
        padding='same'
    ))
    model.add(GlobalAveragePooling1D())
    model.add(Dense(hp.Choice('dense_units', [64, 128]), activation='relu'))
    model.add(Dropout(hp.Choice('dropout', [0.2, 0.3, 0.5])))
    model.add(Dense(1))

    optimizer_name = hp.Choice('optimizer', ['adam', 'rmsprop'])
    learning_rate = hp.Choice('lr', [1e-3, 5e-4, 1e-4])
```

```

if optimizer_name == 'adam':
    optimizer = Adam(learning_rate=learning_rate)
else:
    optimizer = RMSprop(learning_rate=learning_rate)

model.compile(loss='mse', optimizer=optimizer, metrics=['mae'])
return model

tuner_cnn = kt.RandomSearch(
    build_cnn_model,
    objective='val_mae',
    max_trials=10,
    executions_per_trial=1,
    directory='tuner_results',
    project_name='cnn_rul_tuning'
)

tuner_cnn.search(
    X_train_norm, y_train_final,
    validation_data=(X_val_norm, y_val),
    epochs=30,
    batch_size=32,
    verbose=1
)

cnn_trials = tuner_cnn.oracle.get_best_trials(num_trials=10)

print("\n Summary of All CNN Tuning Trials:\n")
for i, trial in enumerate(cnn_trials):
    print(f"Trial {i+1}:")
    print(f" Score (val_mae): {trial.score:.4f}")
    print(" Hyperparameters:")
    for name, value in trial.hyperparameters.values.items():
        print(f" - {name}: {value}")
    print("-" * 40)

```

پارامترهایی را که در ۱۰ Trial بررسی شده اند، به صورت زیر می‌توان نشان داد:

Trial 10 Complete [00h 01m 42s]

val\_mae: 17.903419494628906

Best val\_mae So Far: 12.019194602966309

Total elapsed time: 00h 17m 06s

Summary of All CNN Tuning Trials:

Trial 1:

Score (val\_mae): 12.0192

Hyperparameters:

- filters1: 128
  - filters2: 128
  - dense\_units: 128
  - dropout: 0.5
  - optimizer: adam
  - lr: 0.001
- 

Trial 2:

Score (val\_mae): 12.0911

Hyperparameters:

- filters1: 128
  - filters2: 128
  - dense\_units: 128
  - dropout: 0.3
  - optimizer: rmsprop
  - lr: 0.001
- 

Trial 3:

Score (val\_mae): 12.6621

Hyperparameters:

- filters1: 128
  - filters2: 128
  - dense\_units: 128
  - dropout: 0.5
  - optimizer: rmsprop
  - lr: 0.001
- 

Trial 4:

Score (val\_mae): 12.8410

Hyperparameters:

- filters1: 64
  - filters2: 128
  - dense\_units: 64
  - dropout: 0.5
  - optimizer: adam
  - lr: 0.001
- 

Trial 5:

Score (val\_mae): 12.8921

Hyperparameters:

- filters1: 64
- filters2: 64
- dense\_units: 64

- dropout: 0.2
  - optimizer: rmsprop
  - lr: 0.001
- 

Trial 6:

Score (val\_mae): 13.4374

Hyperparameters:

- filters1: 64
  - filters2: 64
  - dense\_units: 64
  - dropout: 0.5
  - optimizer: rmsprop
  - lr: 0.001
- 

Trial 7:

Score (val\_mae): 14.3927

Hyperparameters:

- filters1: 64
  - filters2: 64
  - dense\_units: 128
  - dropout: 0.2
  - optimizer: rmsprop
  - lr: 0.0005
- 

Trial 8:

Score (val\_mae): 15.5628

Hyperparameters:

- filters1: 64
  - filters2: 64
  - dense\_units: 64
  - dropout: 0.3
  - optimizer: rmsprop
  - lr: 0.0005
- 

Trial 9:

Score (val\_mae): 16.7926

Hyperparameters:

- filters1: 128
  - filters2: 128
  - dense\_units: 128
  - dropout: 0.3
  - optimizer: adam
  - lr: 0.0001
- 

Trial 10:

Score (val\_mae): 17.9034

## Hyperparameters:

- filters1: 128
- filters2: 64
- dense\_units: 128
- dropout: 0.3
- optimizer: rmsprop
- lr: 0.0001

در ادامه، بهترین مقادیر پارامترهای به دست آمده با استفاده از مدل CNN که دارای کمترین میانگین خطای مطلق (MAE) هستند را با کد زیر می‌توان به دست آورد:

```
best_cnn_model = tuner_cnn.get_best_models(1)[0]
best_cnn_hyperparams = tuner_cnn.get_best_hyperparameters(1)[0]

print("Best CNN hyperparameters found:")
for param, val in best_cnn_hyperparams.values.items():
    print(f" - {param}: {val}")
```

بهترین مقادیر پارامترهای به دست آمده با استفاده از مدل CNN بر مبنای حداقل MAE به صورت زیر هستند:

Best CNN hyperparameters found:

- filters1: 128
- filters2: 128
- dense\_units: 128
- dropout: 0.5
- optimizer: adam
- lr: 0.001

نتایج بهترین ترکیب پارامترها بر اساس مدل CNN در جدول زیر ارائه شده است:

Value	Hyperparameter
128	<b>filters1</b>
128	<b>filters2</b>
128	<b>dense_units</b>
0.5	<b>dropout</b>
adam	<b>optimizer</b>
0.001	<b>learning rate</b>
12.019	<b>val_mae</b>

## ۲-۴) تنظیم ابرپارامترها برای شبکه LSTM

برای LSTM نیز ابرپارامترهایی نظیر تعداد واحدهای LSTM، نرخ یادگیری و نوع بهینه‌ساز تنظیم شد تا بهترین پارامترها به منظور به حداقل رساندن خطای MAE روی داده‌های اعتبارسنجی بهدست آید. نرخ یادگیری پایین‌تر (مثلاً 0.0005) در بعضی تنظیمات منجر به همگرایی پایدارتر شد. زیر 0.2 باعث افت دقت شد، در حالی که مقدار 0.3 و 0.5 بهترین تعادل بین دقت و جلوگیری از بیش‌بازش را ایجاد کرد. تنظیمات دقیق منجر به کاهش خطای MAE تا حدود 15 درصد نسبت به تنظیمات اولیه شد.

```
def build_lstm_model(hp):
    model = Sequential()
    model.add(LSTM(
        units=hp.Choice('lstm_units1', [64, 100, 128]),
        return_sequences=True,
        input_shape=(X_train_norm.shape[1], X_train_norm.shape[2])
    ))
    model.add(Dropout(hp.Choice('dropout1', [0.2, 0.3, 0.5])))
    model.add(LSTM(
        units=hp.Choice('lstm_units2', [32, 50, 64]),
        return_sequences=False
    ))
    model.add(Dense(hp.Choice('dense_units', [64, 128]), activation='relu'))
    model.add(Dropout(hp.Choice('dropout2', [0.2, 0.3, 0.5])))
    model.add(Dense(1))

    optimizer_name = hp.Choice('optimizer', ['adam', 'rmsprop'])
    learning_rate = hp.Choice('lr', [1e-3, 5e-4, 1e-4])

    if optimizer_name == 'adam':
        optimizer = Adam(learning_rate=learning_rate)
    else:
        optimizer = RMSprop(learning_rate=learning_rate)

    model.compile(loss='mse', optimizer=optimizer, metrics=['mae'])
    return model

tuner_lstm = kt.RandomSearch(
    build_lstm_model,
    objective='val_mae',
```

```

max_trials=10,
executions_per_trial=1,
directory='tuner_results',
project_name='lstm_rul_tuning'
)

tuner_lstm.search(
    X_train_norm, y_train_final,
    validation_data=(X_val_norm, y_val),
    epochs=30,
    batch_size=32,
    verbose=1
)

lstm_trials = tuner_lstm.oracle.get_best_trials(num_trials=10)

print("\n Summary of All LSTM Tuning Trials:\n")
for i, trial in enumerate(lstm_trials):
    print(f"Trial {i+1}:")
    print(f" Score (val_mae): {trial.score:.4f}")
    print(" Hyperparameters:")
    for name, value in trial.hyperparameters.values.items():
        print(f" - {name}: {value}")
    print("-" * 40)

```

پارامترهایی را که در ۱۰ Trial بررسی شده اند، به صورت زیر می‌توان نشان داد:

Trial 10 Complete [00h 08m 12s]

val\_mae: 9.93221664428711

Best val\_mae So Far: 9.836679458618164

Total elapsed time: 01h 25m 37s

Summary of All LSTM Tuning Trials:

Trial 1:

Score (val\_mae): 9.8367

Hyperparameters:

- lstm\_units1: 64
- dropout1: 0.5
- lstm\_units2: 50
- dense\_units: 128
- dropout2: 0.3
- optimizer: rmsprop
- lr: 0.001

---

Trial 2:

Score (val\_mae): 9.9322

Hyperparameters:

- lstm\_units1: 100
  - dropout1: 0.2
  - lstm\_units2: 32
  - dense\_units: 64
  - dropout2: 0.2
  - optimizer: rmsprop
  - lr: 0.001
- 

Trial 3:

Score (val\_mae): 10.0753

Hyperparameters:

- lstm\_units1: 100
  - dropout1: 0.2
  - lstm\_units2: 50
  - dense\_units: 64
  - dropout2: 0.3
  - optimizer: rmsprop
  - lr: 0.001
- 

Trial 4:

Score (val\_mae): 10.2310

Hyperparameters:

- lstm\_units1: 128
  - dropout1: 0.5
  - lstm\_units2: 32
  - dense\_units: 128
  - dropout2: 0.5
  - optimizer: rmsprop
  - lr: 0.001
- 

Trial 5:

Score (val\_mae): 10.2341

Hyperparameters:

- lstm\_units1: 128
  - dropout1: 0.2
  - lstm\_units2: 64
  - dense\_units: 128
  - dropout2: 0.5
  - optimizer: rmsprop
  - lr: 0.0005
- 

Trial 6:

Score (val\_mae): 10.7565

Hyperparameters:

- lstm\_units1: 100
  - dropout1: 0.5
  - lstm\_units2: 50
  - dense\_units: 64
  - dropout2: 0.5
  - optimizer: rmsprop
  - lr: 0.0005
- 

Trial 7:

Score (val\_mae): 10.8760

Hyperparameters:

- lstm\_units1: 64
  - dropout1: 0.3
  - lstm\_units2: 50
  - dense\_units: 128
  - dropout2: 0.2
  - optimizer: adam
  - lr: 0.0001
- 

Trial 8:

Score (val\_mae): 11.0236

Hyperparameters:

- lstm\_units1: 128
  - dropout1: 0.2
  - lstm\_units2: 32
  - dense\_units: 64
  - dropout2: 0.5
  - optimizer: adam
  - lr: 0.001
- 

Trial 9:

Score (val\_mae): 11.0846

Hyperparameters:

- lstm\_units1: 100
  - dropout1: 0.3
  - lstm\_units2: 50
  - dense\_units: 64
  - dropout2: 0.3
  - optimizer: adam
  - lr: 0.0001
- 

Trial 10:

Score (val\_mae): 11.3580

Hyperparameters:

- lstm\_units1: 100

- dropout1: 0.3
- lstm\_units2: 32
- dense\_units: 64
- dropout2: 0.3
- optimizer: adam
- lr: 0.0001

در ادامه، بهترین مقادیر پارامترهای به دست آمده با استفاده از مدل LSTM که دارای کمترین میانگین خطای

مطلق (MAE) هستند را با کد زیر می‌توان به دست آورد:

```
best_lstm_model = tuner_lstm.get_best_models(1)[0]
best_lstm_hyperparams = tuner_lstm.get_best_hyperparameters(1)[0]

print("Best LSTM hyperparameters found:")
for param, val in best_lstm_hyperparams.values.items():
    print(f" - {param}: {val}")
```

بهترین مقادیر پارامترهای به دست آمده با استفاده از مدل LSTM بر مبنای حداقل MAE به صورت زیر هستند:

Best LSTM hyperparameters found:

- lstm\_units1: 64
- dropout1: 0.5
- lstm\_units2: 50
- dense\_units: 128
- dropout2: 0.3
- optimizer: rmsprop
- lr: 0.001

نتایج بهترین ترکیب پارامترها بر اساس مدل CNN در جدول زیر ارائه شده است:

Value	Hyperparameter
64	<b>lstm_units1</b>
0.5	<b>dropout1</b>
50	<b>lstm_units2</b>
128	<b>dense_units</b>
0.3	<b>dropout2</b>
rmsprop	<b>optimizer</b>
0.001	<b>lr</b>
9.8367	<b>val_mae</b>

## (۵) پیاده‌سازی مدل‌های ترکیبی

### ۱-۵ پیاده‌سازی مدل ترکیبی CNN + LSTM

مدل ترکیبی CNN + LSTM با هدف بهره‌گیری همزمان از قدرت CNN در استخراج ویژگی‌های محلی و توانایی LSTM در یادگیری وابستگی‌های زمانی طراحی شده است. ابتدا، لایه Conv1D با ۶۴ فیلتر و کرنل اندازه ۳ بر روی ورودی اعمال می‌شود تا ویژگی‌های مکانی اولیه استخراج شوند. سپس، لایه MaxPooling1D اندازه CNN داده‌ها را کاهش داده و تمرکز مدل را بر ویژگی‌های مهم افزایش می‌دهد. در مرحله بعد، خروجی لایه‌های Dropout با ۱۰۰ واحد داده می‌شود تا وابستگی‌های زمانی در توالی استخراج گردد. پس از آن یک لایه Dense با ۶۴ نورون شده و نهایتاً خروجی برای جلوگیری از بیش‌برازش استفاده می‌شود. سپس داده‌ها وارد لایه LSTM با ۱۰۰ نورون شده و نهایتاً خروجی مدل از یک نورون برای پیش‌بینی RUL تولید می‌شود. این مدل مانند سایر مدل‌ها باتابع خطای MSE، بهینه‌ساز Adam و معیار ارزیابی MAE آموزش داده شده و با مدل‌های CNN و LSTM مقایسه شده است.

### » طراحی مدل CNN + LSTM

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv1D, MaxPooling1D, LSTM, Dropout, Dense

# Define Input shape
input_shape = (X_train_norm.shape[1], X_train_norm.shape[2])

# Build the CNN + LSTM model
model_cnn_lstm = Sequential([
    Conv1D(filters=64, kernel_size=3, activation='relu', padding='same',
           input_shape=input_shape),
    MaxPooling1D(pool_size=2),
    LSTM(100, return_sequences=False),
    Dropout(0.3),
    Dense(64, activation='relu'),
    Dense(1) # Output: RUL
])

# Compile the CNN + LSTM model
model_cnn_lstm.compile(
    optimizer='adam',
    loss='mse',
    metrics=['mae'])
```

)

```
model_cnn_lstm.summary()
```

نتیجه‌ی طراحی مدل CNN + LSTM را به صورت زیر می‌توان نمایش داد:

**Model: "sequential\_1"**

Layer (type)	Output Shape	Param #
conv1d (Conv1D)	(None, 30, 64)	2,368
max_pooling1d (MaxPooling1D)	(None, 15, 64)	0
lstm_2 (LSTM)	(None, 100)	66,000
dropout_2 (Dropout)	(None, 100)	0
dense_2 (Dense)	(None, 64)	6,464
dense_3 (Dense)	(None, 1)	65

Total params: 74,897 (292.57 KB)

Trainable params: 74,897 (292.57 KB)

Non-trainable params: 0 (0.00 B)

## » آموزش مدل CNN + LSTM

```
# Train the CNN + LSTM model
import time

start_time = time.time()

history_cnn_lstm = model_cnn_lstm.fit(
    X_train_norm, y_train_final,
    validation_data=(X_val_norm, y_val),
    epochs=80,
    batch_size=32,
    verbose=1
)

# Calculate Training Time for CNN + LSTM model
cnn_lstm_train_time = time.time() - start_time
print(f"Training time: {cnn_lstm_train_time:.2f} seconds")
```

زمان آموزش مدل نیز محاسبه شده است که 568.77 ثانیه به دست آمد.

Training time: 568.77 seconds

## ۲-۵) پیاده‌سازی مدل ترکیبی LSTM + CNN

در این مدل، ابتدا وابستگی‌های زمانی توسط لایه LSTM با ۱۰۰ واحد یاد گرفته می‌شود و سپس خروجی آن به لایه Conv1D داده می‌شود تا الگوهای مکانی از خروجی زمانی استخراج شود. این ساختار برای داده‌های مفید است که در آن ویژگی‌های زمانی ابتدا باید تجزیه شوند و سپس از نظر فضایی بررسی گردند. پس از لایه Dropout، یک لایه Conv1D برای کاهش ابعاد و افزایش پایداری مدل اضافه می‌شود. سپس MaxPooling1D و لایه Dense مشابه مدل قبلی استفاده شده و در نهایت یک نورون خروجی مقدار RUL را پیش‌بینی می‌کند. برخلاف مدل CNN + LSTM که ابتدا الگوهای موضعی را یاد می‌گیرد، این مدل ابتدا وابستگی‌های زمانی را استخراج می‌کند و سپس الگوهای فضایی را بررسی می‌کند. مقایسه عملکرد این دو معماری نشان می‌دهد که ترتیب قرارگیری لایه‌ها بر دقت و زمان آموزش اثرگذار است.

### » طراحی مدل LSTM + CNN

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Conv1D, MaxPooling1D, Dropout, Dense,
GlobalAveragePooling1D
from tensorflow.keras.optimizers import Adam

# Define Input shape
input_shape = (X_train_norm.shape[1], X_train_norm.shape[2])

# Build the LSTM + CNN model
model_lstm_cnn = Sequential([
    LSTM(100, return_sequences=True, input_shape=input_shape),
    Dropout(0.3),
    Conv1D(filters=64, kernel_size=3, activation='relu', padding='same'),
    MaxPooling1D(pool_size=2),
    GlobalAveragePooling1D(),
    Dense(64, activation='relu'),
    Dense(1) # Output: RUL
])
# Compile the LSTM + CNN model
model_lstm_cnn.compile(
    optimizer=Adam(learning_rate=0.001),
    loss='mse',
    metrics=['mae'])
```

)

```
model_lstm_cnn.summary()
```

نتیجه‌ی طراحی مدل LSTM + CNN را به صورت زیر می‌توان نمایش داد:

**Model: "sequential\_2"**

Layer (type)	Output Shape	Param #
lstm_3 (LSTM)	(None, 30, 100)	45,200
dropout_3 (Dropout)	(None, 30, 100)	0
conv1d_1 (Conv1D)	(None, 30, 64)	19,264
max_pooling1d_1 (MaxPooling1D)	(None, 15, 64)	0
global_average_pooling1d (GlobalAveragePooling1D)	(None, 64)	0
dense_4 (Dense)	(None, 64)	4,160
dense_5 (Dense)	(None, 1)	65

Total params: 68,689 (268.32 KB)

Trainable params: 68,689 (268.32 KB)

Non-trainable params: 0 (0.00 B)

## » آموزش مدل LSTM + CNN

```
# Train the LSTM + CNN model
```

```
import time
```

```
start_time = time.time()
```

```
history_lstm_cnn = model_lstm_cnn.fit(
    X_train_norm, y_train_final,
    validation_data=(X_val_norm, y_val),
    epochs=80,
    batch_size=32,
    verbose=1
)
```

```
# Calculate Training Time for LSTM + CNN model
```

```
lstm_cnn_train_time = time.time() - start_time
print(f"Training time: {lstm_cnn_train_time:.2f} seconds")
```

زمان آموزش مدل نیز محاسبه شده است که 937.68 ثانیه به دست آمد.

Training time: 937.68 seconds

#### ۶) پیاده‌سازی معماری LSTM + Attention

در این بخش، یک مدل LSTM همراه با سازوکار توجه (Attention) پیادهسازی شده است تا محدودیت‌های LSTM ساده در توجه برابر به تمام بخش‌های داده را برطرف کند. پس از عبور داده‌ها از لایه LSTM، سازوکار توجه به مدل اجازه می‌دهد که به صورت هدفمند بر بخش‌های مهم‌تر توالی تمرکز کند. این کار باعث افزایش دقت پیش‌بینی بهویژه در بخش‌هایی از داده می‌شود که اطلاعات بحرانی درباره نزدیک بودن خرابی دارند. مزایای این مدل شامل بهبود دقت، تفسیرپذیری بالاتر و یادگیری بهتر روابط پیچیده و بلندمدت در داده‌هاست. خروجی Dense Attention به لایه RUL پیش‌بینی می‌شود. در تمرین حاضر، مدل Attention به لایه Dense متصل شده و در نهایت توансته عملکردی بهتر از LSTM ساده داشته باشد، بهویژه در مراحل پایانی عمر موتور که تغییرات سریع‌تری رخ دهد.

LSTM + Attention طراحی مدل ▶

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
from tensorflow.keras.optimizers import Adam

# Define Input shape
input_shape = (X_train_norm.shape[1], X_train_norm.shape[2])

# Build the LSTM + Attention model
model_lstm_att = Sequential([
    LSTM(100, return_sequences=True, input_shape=input_shape),
    Attention(),
    Dropout(0.3),
    Dense(64, activation='relu'),
    Dense(1) # Output: RUL
])

# Compile the LSTM + Attention model
model_lstm_att.compile(
    optimizer=Adam(learning_rate=0.001),
    loss='mse',
    metrics=['mae']
)

model_lstm_att.summary()
```

نتیجه‌ی طراحی مدل LSTM + Attention را به صورت زیر می‌توان نمایش داد:

Model: "sequential\_3"

Layer (type)	Output Shape	Param #
lstm_4 (LSTM)	(None, 30, 100)	45,200
attention (Attention)	(None, 100)	0
dropout_4 (Dropout)	(None, 100)	0
dense_6 (Dense)	(None, 64)	6,464
dense_7 (Dense)	(None, 1)	65

Total params: 51,729 (202.07 KB)

Trainable params: 51,729 (202.07 KB)

Non-trainable params: 0 (0.00 B)

## » آموزش مدل LSTM + Attention

```
# Train the LSTM + Attention model
import time

start_time = time.time()

history_lstm_att = model_lstm_att.fit(
    X_train_norm, y_train_final,
    validation_data=(X_val_norm, y_val),
    epochs=80,
    batch_size=32,
    verbose=1
)

# Calculate Training Time for LSTM + Attention model
lstm_att_train_time = time.time() - start_time
print(f"Training time: {lstm_att_train_time:.2f} seconds")
```

زمان آموزش مدل نیز محاسبه شده است که 811.84 ثانیه به دست آمد.

Training time: 811.84 seconds

## بخش ب) ارزیابی مدل‌ها

### (۱) نمودارها و مقایسه بصری

برای بررسی عملکرد مدل‌ها، سه نمودار برای هر مدل رسم شده است. نمودارهای خطای میانگین مربعات (MSE) و خطای میانگین مطلق (MAE) برای هر دوره‌ی آموزش و اعتبارسنجی رسم شد تا روند یادگیری و احتمال بیش‌بازش (Overfitting) مشخص شود. همچنین، برای هر مدل، نمودار مقایسه مقدار پیش‌بینی شده RUL در برابر مقدار واقعی روی داده‌های تست رسم شده است. در این نمودارها، مدل‌های ترکیبی و Attention عملکرد بهتری در همترازی نقاط واقعی نشان دادند. این مقایسه بصری نقش مهمی در تشخیص ضعف مدل‌ها در بخش‌های خاص از چرخه عمر موتور دارد. به عنوان مثال، مدل LSTM + Attention در مراحل انتهایی عمر موتور دقیق‌تری دارد که در نمودارها به خوبی مشهود است.

❖ رسم نمودارهای MSE و MAE بر حسب ایپاک برای هر مدل

✓ رسم نمودارهای MSE و MAE برای مدل CNN

```
# Plot MSE (Mean Squared Error) & MAE (Mean Absolute Error) over epochs for CNN model
```

```
plt.figure(figsize=(12, 4), dpi=300)
```

```
# MSE
```

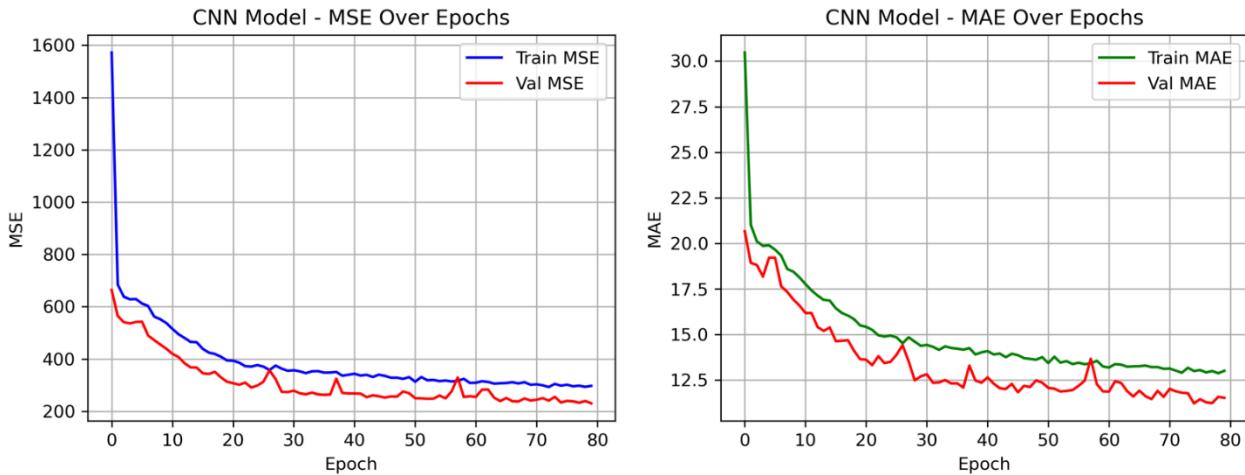
```
plt.subplot(1, 2, 1)
plt.plot(history_cnn.history['loss'], label='Train MSE', color='blue')
plt.plot(history_cnn.history['val_loss'], label='Val MSE', color='red')
plt.title('CNN Model - MSE Over Epochs', fontsize=12)
plt.xlabel('Epoch', fontsize=10)
plt.ylabel('MSE', fontsize=10)
plt.legend()
plt.grid(True)
```

```
# MAE
```

```
plt.subplot(1, 2, 2)
plt.plot(history_cnn.history['mae'], label='Train MAE', color='green')
plt.plot(history_cnn.history['val_mae'], label='Val MAE', color='red')
plt.title('CNN Model - MAE Over Epochs', fontsize=12)
plt.xlabel('Epoch', fontsize=10)
plt.ylabel('MAE', fontsize=10)
plt.legend()
```

```
plt.grid(True)
```

```
plt.subplots_adjust(wspace=0.2)
plt.savefig(r'G:\Artificial Intelligence\Tamrin\Tamrin5\Results\MSE&MAE_CNN.png')
plt.show()
```



شکل ۲: نمودارهای MSE و MAE برای مدل CNN

### ✓ رسم نمودارهای MSE و MAE برای مدل LSTM

```
# Plot MSE (Mean Squared Error) & MAE (Mean Absolute Error) over epochs for LSTM model
```

```
plt.figure(figsize=(12, 4), dpi=300)
```

#### # MSE

```
plt.subplot(1, 2, 1)
plt.plot(history_lstm.history['loss'], label='Train MSE', color='blue')
plt.plot(history_lstm.history['val_loss'], label='Val MSE', color='red')
plt.title('LSTM Model - MSE Over Epochs', fontsize=12)
plt.xlabel('Epoch', fontsize=10)
plt.ylabel('MSE', fontsize=10)
plt.legend()
plt.grid(True)
```

#### # MAE

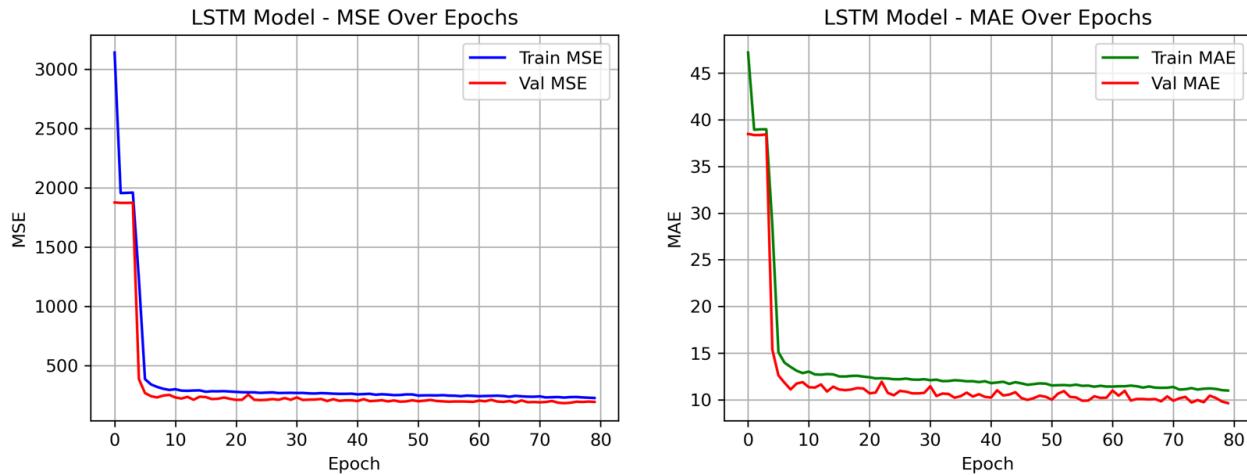
```
plt.subplot(1, 2, 2)
plt.plot(history_lstm.history['mae'], label='Train MAE', color='green')
plt.plot(history_lstm.history['val_mae'], label='Val MAE', color='red')
plt.title('LSTM Model - MAE Over Epochs', fontsize=12)
plt.xlabel('Epoch', fontsize=10)
plt.ylabel('MAE', fontsize=10)
```

```

plt.legend()
plt.grid(True)

plt.subplots_adjust(wspace=0.2)
plt.savefig(r'G:\Artificial_Intelligence\Tamrin\tamrin5\Results\MSE&MAE_LSTM.png')
plt.show()

```



شکل ۳: نمودارهای MSE و MAE برای مدل LSTM

### ✓ رسم نمودارهای MSE و MAE برای مدل CNN + LSTM

```
# Plot MSE (Mean Squared Error) & MAE (Mean Absolute Error) over epochs for CNN +
LSTM model
```

```

plt.figure(figsize=(12, 4), dpi=300)

# MSE
plt.subplot(1, 2, 1)
plt.plot(history_cnn_lstm.history['loss'], label='Train MSE', color='blue')
plt.plot(history_cnn_lstm.history['val_loss'], label='Val MSE', color='red')
plt.title('CNN + LSTM Model - MSE Over Epochs', fontsize=12)
plt.xlabel('Epoch', fontsize=10)
plt.ylabel('MSE', fontsize=10)
plt.legend()
plt.grid(True)

# MAE
plt.subplot(1, 2, 2)
plt.plot(history_cnn_lstm.history['mae'], label='Train MAE', color='green')
plt.plot(history_cnn_lstm.history['val_mae'], label='Val MAE', color='red')
plt.title('CNN + LSTM Model - MAE Over Epochs', fontsize=12)

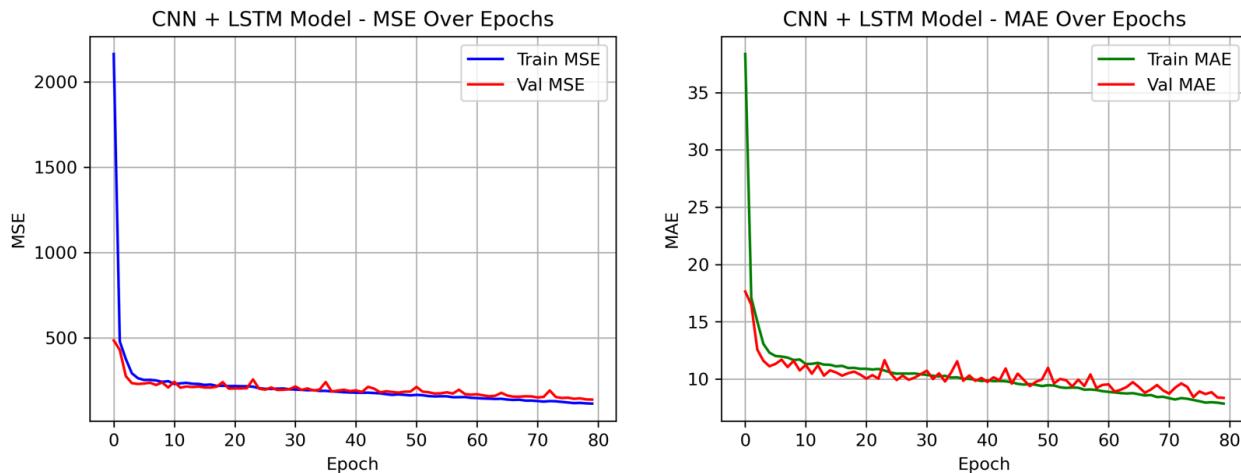
```

```

plt.xlabel('Epoch', fontsize=10)
plt.ylabel('MAE', fontsize=10)
plt.legend()
plt.grid(True)

plt.subplots_adjust(wspace=0.2)
plt.savefig(r'G:\Artificial_ Intelligence\Tamrin\Tamrin5\Results\MSE&MAE_CNN+LSTM.png')
plt.show()

```



شکل ۴: نمودارهای MSE و MAE برای مدل CNN + LSTM

#### ✓ رسم نمودارهای MSE و MAE برای مدل LSTM + CNN

```

# Plot MSE (Mean Squared Error) & MAE (Mean Absolute Error) over epochs for LSTM +
# CNN model

plt.figure(figsize=(12, 4), dpi=300)

# MSE
plt.subplot(1, 2, 1)
plt.plot(history_lstm_cnn.history['loss'], label='Train MSE', color='blue')
plt.plot(history_lstm_cnn.history['val_loss'], label='Val MSE', color='red')
plt.title('LSTM + CNN Model - MSE Over Epochs', fontsize=12)
plt.xlabel('Epoch', fontsize=10)
plt.ylabel('MSE', fontsize=10)
plt.legend()
plt.grid(True)

# MAE
plt.subplot(1, 2, 2)
plt.plot(history_lstm_cnn.history['mae'], label='Train MAE', color='green')

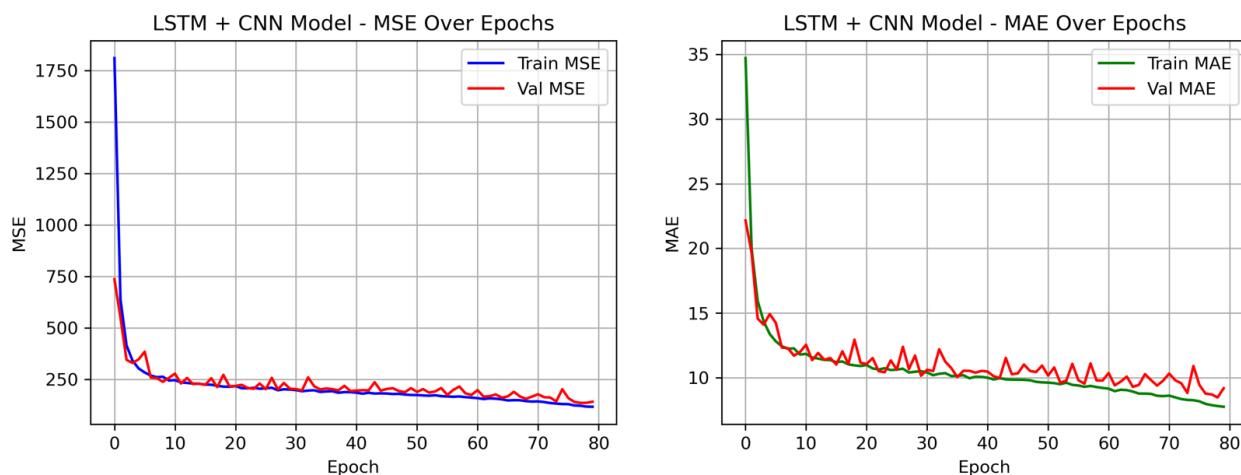
```

```

plt.plot(history_lstm_cnn.history['val_mae'], label='Val MAE', color='red')
plt.title('LSTM + CNN Model - MAE Over Epochs', fontsize=12)
plt.xlabel('Epoch', fontsize=10)
plt.ylabel('MAE', fontsize=10)
plt.legend()
plt.grid(True)

plt.subplots_adjust(wspace=0.2)
plt.savefig(r'G:\Artificial_Intelligence\Tamrin\Tamrin5\Results\MSE&MAE_LSTM+CNN.png')
plt.show()

```



شکل ۵: نمودارهای MSE و MAE برای مدل LSTM + CNN

### ✓ رسم نمودارهای MSE و MAE برای مدل LSTM + Attention

```
# Plot MSE (Mean Squared Error) & MAE (Mean Absolute Error) over epochs for LSTM + Attention model
```

```
plt.figure(figsize=(12, 4), dpi=300)
```

```
# MSE
```

```
plt.subplot(1, 2, 1)
plt.plot(history_lstm_att.history['loss'], label='Train MSE', color='blue')
plt.plot(history_lstm_att.history['val_loss'], label='Val MSE', color='red')
plt.title('LSTM + Attention Model - MSE Over Epochs', fontsize=12)
plt.xlabel('Epoch', fontsize=10)
plt.ylabel('MSE', fontsize=10)
plt.legend()
plt.grid(True)
```

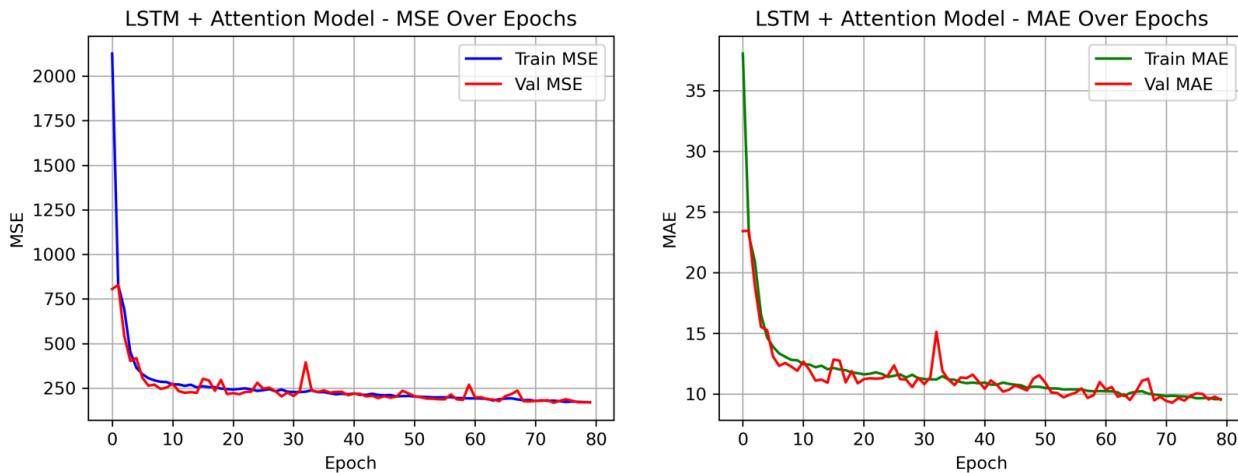
```
# MAE
```

```

plt.subplot(1, 2, 2)
plt.plot(history_lstm_att.history['mae'], label='Train MAE', color='green')
plt.plot(history_lstm_att.history['val_mae'], label='Val MAE', color='red')
plt.title('LSTM + Attention Model - MAE Over Epochs', fontsize=12)
plt.xlabel('Epoch', fontsize=10)
plt.ylabel('MAE', fontsize=10)
plt.legend()
plt.grid(True)

plt.subplots_adjust(wspace=0.2)
plt.savefig(r'G:\Artificial_Intelligence\Tamrin\tamrin5\Results\MSE&MAE_LSTM+Attention.png')
plt.show()

```



شکل ۶: نمودارهای MSE و MAE برای مدل LSTM + Attention

❖ رسم نمودار مقادیر پیش‌بینی شده RUL در برابر مقدار واقعی روی داده‌های تست برای هر مدل

✓ نمودار مقادیر پیش‌بینی شده RUL در برابر مقدار واقعی برای مدل CNN

```
# Plot Real VS Predicted RUL for CNN model
import matplotlib.pyplot as plt
```

```

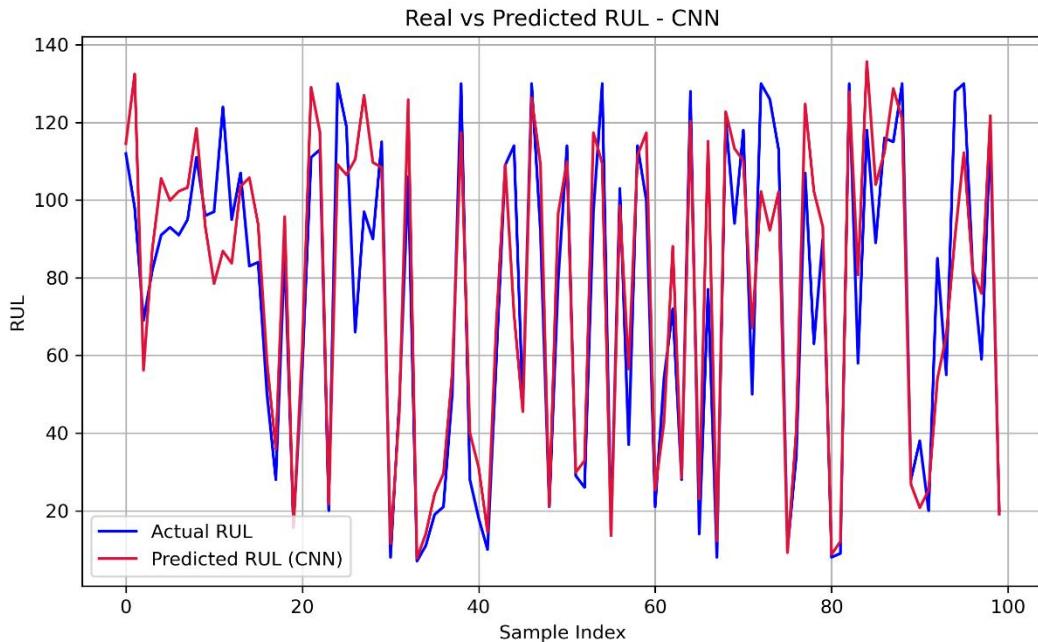
plt.figure(figsize=(8, 5), dpi=300)
plt.plot(y_test, label='Actual RUL', color='blue')
plt.plot(model_cnn.predict(X_test_norm).flatten(), label='Predicted RUL (CNN)',
color='crimson')
plt.title('Real vs Predicted RUL - CNN', fontsize=12)
plt.xlabel('Sample Index', fontsize=10)
plt.ylabel('RUL', fontsize=10)
plt.legend()

```

```

plt.grid(True)
plt.tight_layout()
plt.savefig(r'G:\Artificial_Intelligence\Tamrin\tamrin5\Results\Real_VS_Predicted_RUL_CNN.
png')
plt.show()

```



شکل ۷: نمودار مقادیر پیش‌بینی شده RUL در برابر مقدار واقعی برای مدل CNN

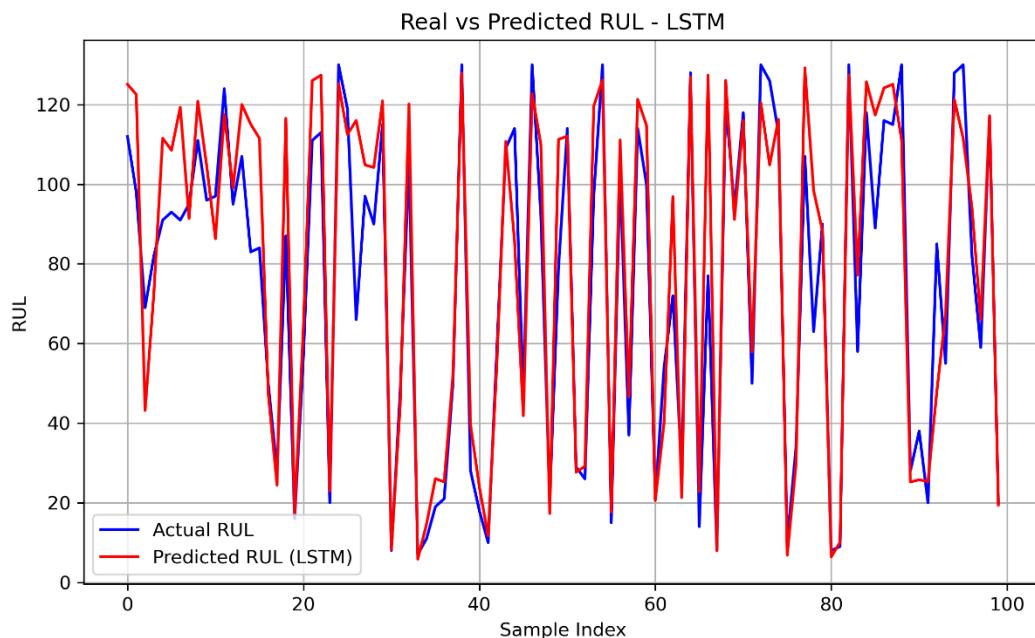
#### ✓ نمودار مقادیر پیش‌بینی شده RUL در برابر مقدار واقعی برای مدل LSTM

```
# Plot Real VS Predicted RUL for LSTM model
```

```

plt.figure(figsize=(8, 5), dpi=300)
plt.plot(y_test, label='Actual RUL', color='blue')
plt.plot(model_lstm.predict(X_test_norm).flatten(), label='Predicted RUL (LSTM)', color='red')
plt.title('Real vs Predicted RUL - LSTM', fontsize=12)
plt.xlabel('Sample Index', fontsize=10)
plt.ylabel('RUL', fontsize=10)
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.savefig(r'G:\Artificial_Intelligence\Tamrin\tamrin5\Results\Real_VS_Predicted_RUL_LST
M.png')
plt.show()

```

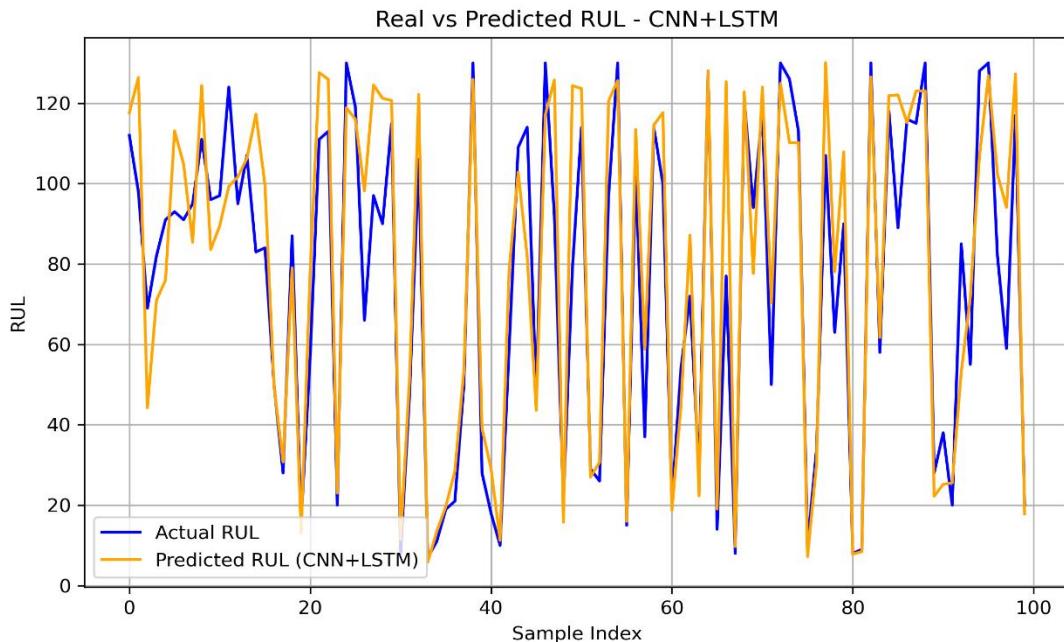


شکل ۸: نمودار مقادیر پیش‌بینی شده RUL در برابر مقدار واقعی برای مدل LSTM

✓ نمودار مقادیر پیش‌بینی شده RUL در برابر مقدار واقعی برای مدل CNN + LSTM

```
# Plot Real VS Predicted RUL for CNN + LSTM model
```

```
plt.figure(figsize=(8, 5), dpi=300)
plt.plot(y_test, label='Actual RUL', color='blue')
plt.plot(model_cnn_lstm.predict(X_test_norm).flatten(), label='Predicted RUL (CNN+LSTM)', color='orange')
plt.title('Real vs Predicted RUL - CNN+LSTM', fontsize=12)
plt.xlabel('Sample Index', fontsize=10)
plt.ylabel('RUL', fontsize=10)
plt.legend(loc='lower left', bbox_to_anchor=(0.0, 0.0), fancybox=True, shadow=False)
plt.grid(True)
plt.tight_layout()
plt.savefig(r'G:\Artificial_Intelligence\Tamrin\Tamrin5\Results\Real_VS_Predicted_RUL_CNN+LSTM.png')
plt.show()
```

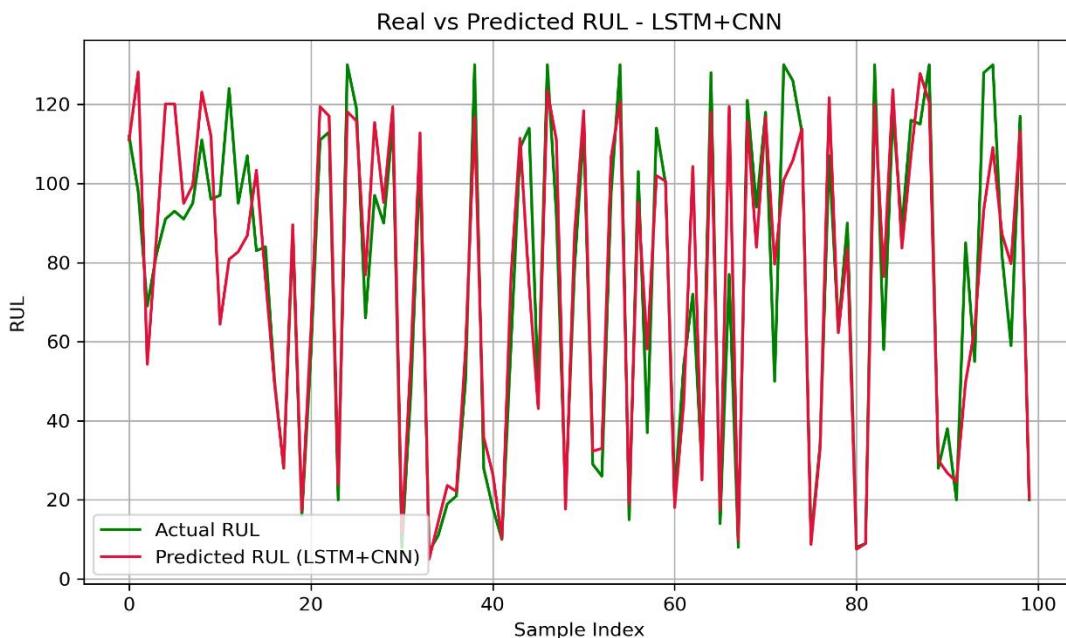


شکل ۹: نمودار مقادیر پیش‌بینی شده RUL در برابر مقدار واقعی برای مدل CNN + LSTM

✓ نمودار مقادیر پیش‌بینی شده RUL در برابر مقدار واقعی برای مدل LSTM + CNN

```
# Plot Real VS Predicted RUL for LSTM + CNN model
```

```
plt.figure(figsize=(8, 5), dpi=300)
plt.plot(y_test, label='Actual RUL', color='green')
plt.plot(model_lstm_cnn.predict(X_test_norm).flatten(), label='Predicted RUL (LSTM+CNN)', color='crimson')
plt.title('Real vs Predicted RUL - LSTM+CNN', fontsize=12)
plt.xlabel('Sample Index', fontsize=10)
plt.ylabel('RUL', fontsize=10)
plt.legend(loc='lower left', bbox_to_anchor=(0.0, 0.0), fancybox=True, shadow=False)
plt.grid(True)
plt.tight_layout()
plt.savefig(r'G:\Artificial_Intelligence\Tamrin\tamrin5\Results\Real_VS_Predicted_RUL_LSTM+CNN.png')
plt.show()
```

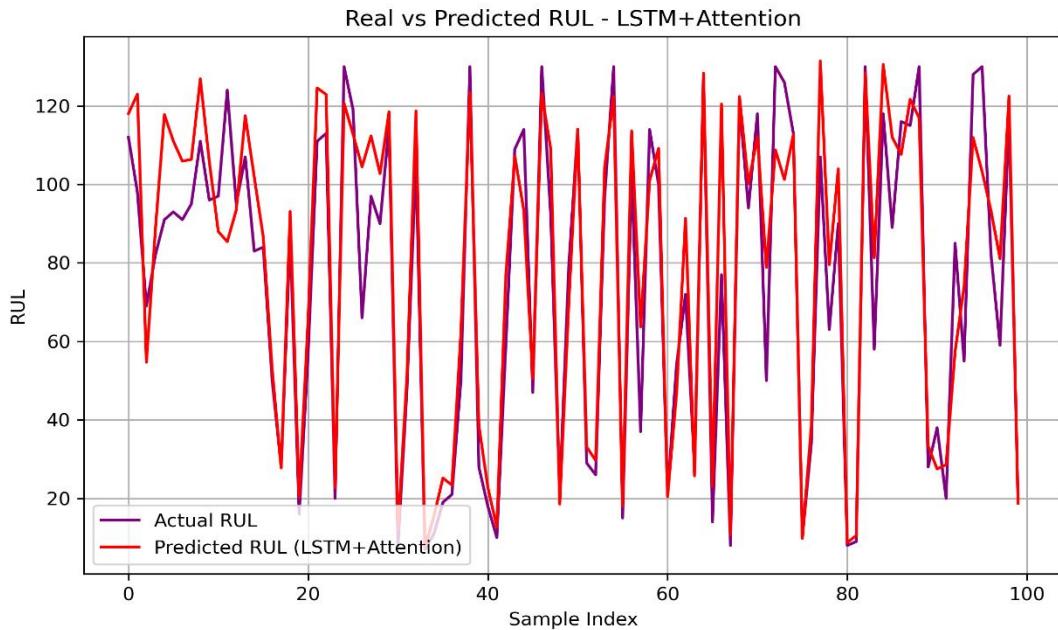


شکل ۱۰: نمودار مقادیر پیش‌بینی شده RUL در برابر مقدار واقعی برای مدل LSTM + CNN

✓ نمودار مقادیر پیش‌بینی شده RUL در برابر مقدار واقعی برای مدل LSTM + Attention

```
# Plot Real VS Predicted RUL for LSTM + Attention model
```

```
plt.figure(figsize=(8, 5), dpi=300)
plt.plot(y_test, label='Actual RUL', color='purple')
plt.plot(model_lstm_att.predict(X_test_norm).flatten(), label='Predicted RUL (LSTM+Attention)', color='red')
plt.title('Real vs Predicted RUL - LSTM+Attention', fontsize=12)
plt.xlabel('Sample Index', fontsize=10)
plt.ylabel('RUL', fontsize=10)
plt.legend(loc='lower left', bbox_to_anchor=(0.0, 0.0), fancybox=True, shadow=False)
plt.grid(True)
plt.tight_layout()
plt.savefig(r'G:\Artificial_Intelligence\Tamrin\Tamrin5\Results\Real_VS_Predicted_RUL_LSTM+Attention.png')
plt.show()
```



شکل ۱۱: نمودار مقادیر پیش‌بینی شده RUL در برابر مقدار واقعی برای مدل LSTM + Attention

## ۲) معیارهای ارزیابی مدل

برای هر مدل، معیارهای MAE، RMSE، R<sup>2</sup> و زمان آموزش با استفاده از کد زیر محاسبه می‌شود:

```
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import numpy as np
import time

# Define helper function
def evaluate_model(y_true, y_pred, training_time):
    mae = mean_absolute_error(y_true, y_pred)
    rmse = np.sqrt(mean_squared_error(y_true, y_pred))
    r2 = r2_score(y_true, y_pred)
    mape = np.mean(np.abs((y_true - y_pred) / np.maximum(y_true, 1))) * 100 # Avoid divide-by-zero
    return {
        'MAE': round(mae, 2),
        'RMSE': round(rmse, 2),
        'R2': round(r2, 4),
        'MAPE (%)': round(mape, 2),
        'Train Time (s)': round(training_time, 2)
    }

# Training times
```

```

train_times = {
    'CNN': cnn_train_time,
    'LSTM': lstm_train_time,
    'CNN+LSTM': cnn_lstm_train_time,
    'LSTM+CNN': lstm_cnn_train_time,
    'LSTM+Attention': lstm_att_train_time
}

# Predict on test set
preds = {
    'CNN': model_cnn.predict(X_test_norm).flatten(),
    'LSTM': model_lstm.predict(X_test_norm).flatten(),
    'CNN+LSTM': model_cnn_lstm.predict(X_test_norm).flatten(),
    'LSTM+CNN': model_lstm_cnn.predict(X_test_norm).flatten(),
    'LSTM+Attention': model_lstm_att.predict(X_test_norm).flatten()
}

# Evaluate all models
results = {}
for name, y_pred in preds.items():
    results[name] = evaluate_model(y_test, y_pred, train_times[name])

# Display results as DataFrame
import pandas as pd
results_df = pd.DataFrame(results).T # Transpose for readability
print("Model Evaluation Results:")
print(results_df)

```

که نتایج آن به صورت زیر است:

#### Model Evaluation Results:

	MAE	RMSE	R <sup>2</sup>	MAPE (%)	Train Time (s)
CNN	11.88	15.99	0.8455	18.99	376.76
LSTM	10.88	15.25	0.8595	16.89	1525.29
CNN+LSTM	11.87	16.05	0.8444	18.59	568.77
LSTM+CNN	10.52	14.76	0.8684	15.90	937.68
LSTM+Attention	10.77	14.18	0.8786	17.55	811.84

## جدول نتایج ارزیابی مدل‌ها:

مدل	MAE	RMSE	$R^2$	MAPE (%)	زمان آموزش (ثانیه)
CNN	11.88	15.99	0.8455	18.99	376.76
LSTM	10.88	15.25	0.8595	16.89	1525.29
CNN + LSTM	11.87	16.05	0.8444	18.59	568.77
LSTM + CNN	10.52	14.76	0.8684	15.90	937.68
LSTM + Attention	10.77	14.18	0.8786	17.55	811.84

## (۳) تحلیل عملکرد (خلاصه)

۱. کدام مدل دقیق‌ترین پیش‌بینی را ارائه داد؟ دلیل عملکرد بهتر آن چه بود؟

دقیق‌ترین مدل: LSTM + Attention

$$\text{بالاترین مقدار } R^2 = 0.8786$$

$$\text{کمترین مقدار } RMSE = 14.18$$

LSTM به خوبی برای یادگیری دنباله‌های زمانی مناسب است. مکانیسم Attention به مدل اجازه می‌دهد به بخش‌های مهم‌تر از دنباله تمرکز کند و به جای وابستگی یکسان به همه زمان‌ها، توجه مدل را روی زمان‌های کلیدی معطوف کند. ترکیب این دو ویژگی، دقت مدل را افزایش داده است.

۲. آیا هیچ یک از مدل‌ها نشانه‌هایی از بیش‌برازش (Overfitting) داشتند؟ چگونه متوجه شدید؟

بله، مدل CNN + LSTM نشانه‌هایی از بیش‌برازش دارد. زیرا خطای آموزش کمتر از اعتبارسنجی (val) در نمودارهای MSE و MAE مشاهده می‌شود. هم‌چنان، زمان آموزش متوسط (568.77) با دقت ضعیف:  $R^2 = 0.8444$  دارد. علت می‌تواند پیچیدگی مدل نسبت به حجم داده یا مقدار dropout ناکافی باشد.

### ۳. درباره تعادل بین دقت (Accuracy) و زمان آموزش بحث کنید.

LSTM + Attention، بهترین دقت ( $R^2 = 0.8786$ )، زمان آموزش ۸۱۱.۸۴، تعادل عالی.

LSTM + CNN، دومین دقت خوب ( $R^2 = 0.8684$ )، زمان آموزش ۹۳۷.۶۸، متعادل.

LSTM، خوب ولی زمان زیاد ( $R^2 = 0.8595$ )، زمان آموزش ۱۵۲۵.۲۹، دقت بالا ولی هزینه زمانی بالا.

CNN، سریع‌ترین مدل ولی کم‌دقت‌ترین ( $R^2 = 0.8455$ )، زمان آموزش ۷۶.۷۶، زمان عالی، دقت نه‌چندان خوب.

✓ مدل‌هایی مثل LSTM + Attention یا LSTM+CNN تعادل خوبی بین دقت و زمان برقرار کردند.

✓ اگر سرعت مهم‌تر از دقت باشد، مدل CNN یا LSTM + CNN انتخاب مناسبی است.

✓ اگر دقت اهمیت دارد، LSTM + Attention انتخاب بهتری است حتی با صرف زمان کمی بیشتر.

بر اساس نتایج، مدل LSTM + Attention بهترین عملکرد را از نظر دقت پیش‌بینی ارائه داد. توانایی آن در تمرکز بر بخش‌های مهم داده باعث شد وابستگی‌های بلندمدت و نوسانات حیاتی در داده‌ها بهتر مدل‌سازی شوند. مدل CNN + LSTM نیز ترکیب خوبی از دقت و سرعت داشت. در مقابل، مدل CNN با وجود سادگی و سرعت بالا، دقت کمتری در مراحل انتهایی عمر موتور داشت. در برخی مدل‌ها مانند LSTM ساده، نشانه‌هایی از بیش‌برازش دیده شد که با افزایش خطای اعتبارسنجی در دوره‌های انتهایی مشخص شد. در بحث تعادل دقت و زمان، مدل‌های ترکیبی با ساختارهای مناسب (مانند CNN + LSTM) توانستند نتایجی متعادل ارائه دهند، در حالی‌که مدل Attention با وجود دقت بالا، به زمان آموزش بیشتری نیاز داشت. در نهایت، انتخاب مدل باید با توجه به محدودیت‌های زمانی، منابع پردازشی و نیاز به دقت انجام شود.

### نتیجه‌گیری

در این تمرین، مسئله پیش‌بینی Remaining Useful Life (RUL) برای موتورهای توربوفن با استفاده از داده‌های سری زمانی از مجموعه CMAPSS بررسی شد. با طی مراحل منظم شامل پاکسازی و آماده‌سازی

داده‌ها، برچسب‌گذاری RUL، نرمال‌سازی و تقسیم‌بندی با پنجره لغزان، بستر مناسبی برای آموزش مدل‌های یادگیری عمیق فراهم شد. ابتدا دو مدل پایه شامل CNN و LSTM پیاده‌سازی و مقایسه شدند که هر یک مزایا LSTM + CNN + LSTM و CNN + Attention را نشان دادند. سپس، با توسعه مدل‌های ترکیبی مانند LSTM + CNN + سعی شد از قابلیت‌های مشترک دو معماری بهره‌برداری شود. در گام نهایی، مدل طراحی شد که با بهره‌گیری از سازوکار Attention، توانست تمرکز بیشتری بر بخش‌های بحرانی داده داشته باشد و در نتیجه T دقت بالاتری در پیش‌بینی RUL ارائه داد.

نتایج به‌دست‌آمده نشان داد که مدل LSTM + Attention از نظر MAE، RMSE و  $R^2$  بهترین عملکرد را داشت و توانست رفتار سیستم را در مراحل پایانی عمر به خوبی مدل‌سازی کند. مدل CNN با وجود زمان آموزش بسیار کم، دقت پایین‌تری نسبت به سایر مدل‌ها داشت و بیشتر در مراحل اولیه عمر موتور قابل اعتماد بود. در حالی که مدل‌های ترکیبی تعادلی مناسب بین دقت و زمان آموزش برقرار کردند. تحلیل منحنی خطأ و نمودارهای مقایسه‌ای نیز این یافته‌ها را تأیید کردند. در برخی موارد، نشانه‌هایی از بیش‌برازش در مدل‌هایی با تعداد پارامتر زیاد مشاهده شد که با به‌کارگیری Dropout و تنظیم ابرپارامترها تا حد زیادی کنترل شد.

در مجموع، استفاده از مدل‌های ترکیبی نه تنها باعث افزایش دقت مدل شد، بلکه پیش‌بینی‌ها را نیز بهبود داد. این رویکردها در کاربردهای صنعتی واقعی که نیاز به تصمیم‌گیری دقیق در نگهداری دارند، می‌توانند نقش مهمی ایفا کنند.

## گیت هاب

گزارش و کد مربوط به این تمرین در گیت هاب به آدرس

[https://github.com/MM-Touiserkani/AI\\_HW5\\_Touiserkani](https://github.com/MM-Touiserkani/AI_HW5_Touiserkani)

قرار داده شده است.