|  |  |  |
| --- | --- | --- |
| **پرسش ۱** | **نام و نام خانوادگی** | سیدرضا مسلمی |
| **شماره دانشجویی** | 810103326 |
| **پرسش ۲** | **نام و نام خانوادگی** | BIBI RUQIA |
| **شماره دانشجویی** | 810102053 |
|  | **مهلت ارسال پاسخ** | **۱۴۰3.10.13** |

|  |  |  |
| --- | --- | --- |
|  | **به نام خدا**  **دانشگاه تهران**  **دانشکده‌ مهندسی برق و کامپیوتر** |  |
| **درس شبکه‌های عصبی و یادگیری عمیق**  **تمرین پنجم** | | |

**فهرست**

[**قوانین** 1](#_Toc187167606)

[**پرسش 1**. **عنوان پرسش اول به فارسی** 1](#_Toc187167607)

[۱-۱. مقدمه 1](#_Toc187167608)

[۱-2. آماده‌سازی 4](#_Toc187167609)

[۱-3. روش‌شناسی و نتایج 11](#_Toc187167610)

[**پرسش ۲** **-** استفاده از **ViT**برای طبقهبندی تصاویر گلبولهای سفید 18](#_Toc187167611)

[۱-۲.مقدمه 18](#_Toc187167612)

[1. Accuracy Plot (Left Panel): 26](#_Toc187167613)

[2. Loss Plot (Right Panel): 27](#_Toc187167614)

[Key Insights: 27](#_Toc187167615)

[1. Accuracy Plot (Left Panel): 31](#_Toc187167616)

[2. Loss Plot (Right Panel): 31](#_Toc187167617)

[Key Takeaways: 31](#_Toc187167618)

[Key Insights 33](#_Toc187167619)

[Key Insights: 36](#_Toc187167620)

[Key Insights: 38](#_Toc187167621)

[Comparison of CNN and ViT Performance 41](#_Toc187167622)

[Summary: 41](#_Toc187167623)

# **قوانین**

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

* از پاسخ‌های خود یک گزارش در قالبی که در صفحه‌ی درس در سامانه‌ی Elearn با نام ***REPORTS\_TEMPLATE.docx*** قرار داده شده تهیه نمایید.
* پیشنهاد می‌شود تمرین‌ها را در قالب گروه‌های دو نفره انجام دهید. (بیش از دو نفر مجاز نیست و تحویل تک نفره نیز نمره‌ی اضافی ندارد) توجه نمایید الزامی در یکسان ماندن اعضای گروه تا انتهای ترم وجود ندارد. (یعنی، می‌توانید تمرین اول را با شخص A و تمرین‌ دوم را با شخص B و ... انجام دهید)
* **کیفیت گزارش شما در فرآيند تصحيح از اهميت ويژه­اي برخوردار است**؛ بنابراین، لطفا تمامی نکات و فرض­هایی را كه در پیاده­سازی­ها و محاسبات خود در نظر مي­گيريد در گزارش ذکر کنید.
* در گزارش خود مطابق با آنچه در قالب نمونه قرار داده شده، برای شکل‌ها زیرنویس و برای جدول‌ها بالانویس در نظر بگیرید.
* الزامی به ارائه توضیح جزئیات کد در گزارش نیست، اما باید نتایج بدست آمده از آن را گزارش و تحلیل کنید.
* **تحلیل نتایج الزامی می‌باشد، حتی اگر در صورت پرسش اشاره‌ای به آن نشده باشد.**
* **دستیاران آموزشی ملزم به اجرا کردن کدهای شما نیستند**؛ بنابراین، هرگونه نتیجه و یا تحلیلی که در صورت پرسش از شما خواسته شده را به طور واضح و کامل در گزارش بیاورید. در صورت عدم رعایت این مورد، بدیهی است که از نمره تمرین کسر می­شود.
* **کدها حتما باید در قالب نوت‌بوک با پسوند .ipynb تهیه شوند، در پایان کار، تمامی کد اجرا شود و خروجی هر سلول حتما در این فایل ارسالی شما ذخیره شده باشد.** بنابراین برای مثال اگر خروجی سلولی یک نمودار است که در گزارش آورده‌اید، این نمودار باید هم در گزارش هم در نوت‌بوک کد‌ها وجود داشته باشد.
* **در صورت مشاهده‌ی تقلب امتیاز تمامی افراد شرکت­کننده در آن، 100- لحاظ می­شود.**
* تنها زبان برنامه نویسی مجاز **Python** است.
* **استفاده از کدهای آماده برای تمرین­ها به­ هیچ ­وجه مجاز نیست. در صورتی که دو گروه از یک منبع مشترک استفاده کنند و کدهای مشابه تحویل دهند، تقلب محسوب می‌شود.**
* نحوه محاسبه­ تاخیر به این شکل است: پس از پایان رسیدن مهلت ارسال گزارش، حداکثر تا یک هفته امکان ارسال با تاخیر وجود دارد، پس از این یک هفته نمره آن تکلیف برای شما صفر خواهد شد.
  + سه روز اول: بدون جریمه
  + روز چهارم: ۵ درصد
  + روز پنجم: ۱۰ درصد
  + روز ششم: ۱۵ درصد
  + روز هفتم: ۲۰ درصد
* حداکثر نمره‌ای که برای هر سوال می‌توان اخد کرد ۱۰۰ بوده و اگر مجموع بارم یک **سوال** بیشتر از ۱۰۰ باشد، در صورت اخد نمره بیشتر از ۱۰۰، اعمال نخواهد شد.
  + برای مثال: اگر نمره اخذ شده از سوال ۱ برابر ۱۰۵ و نمره سوال ۲ برابر ۹۵ باشد، نمره نهایی تمرین ۹۷.۵ خواهد بود و نه ۱۰۰.
* لطفا گزارش، کدها و سایر ضمایم را به در یک پوشه با نام زیر قرار داده و آن را فشرده سازید، سپس در سامانه‌ی Elearn بارگذاری نمایید:

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

(مثال: HW1\_Ahmadi\_810199101\_Bagheri\_810199102.zip)

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

# **پرسش 1**. **عنوان پرسش اول به فارسی**

۱-۱. مقدمه

**۱. محدودیت‌های روش‌های آماری سنتی مانند ARIMA و GARCH چیست؟**

سازگاری محدود: روش‌های آماری سنتی مانند ARIMA و GARCH بر فرضیات سخت‌گیرانه‌ای درباره‌ی توزیع داده‌ها و هموار بودن آن‌ها تکیه دارند که توانایی این مدل‌ها را در انطباق با شرایط و مجموعه داده‌های مختلف کاهش می‌دهد.

ناتوانی در مدل‌سازی وابستگی‌های غیرخطی: این روش‌ها عمدتاً روابط خطی را مدل‌سازی می‌کنند و قادر به درک وابستگی‌های پیچیده، غیرخطی و تصادفی در داده‌های توان بادی فراساحلی نیستند.

عملکرد ضعیف در پیش‌بینی‌های بلندمدت: روش‌های آماری به دلیل محدودیت در درک وابستگی‌های بلندمدت در داده‌های سری زمانی، در پیش‌بینی‌های بلندمدت با مشکل مواجه هستند.

**۲. برخی از مزایای مدل‌های یادگیری ماشین مانند SVM، Random Forest و XGBoost نسبت به روش‌های آماری سنتی در پیش‌بینی توان بادی را ذکر کنید.**

مدیریت بهتر روابط غیرخطی: مدل‌های یادگیری ماشین می‌توانند وابستگی‌های پیچیده و غیرخطی را در داده‌های توان بادی که روش‌های آماری سنتی قادر به شناسایی آن‌ها نیستند، تشخیص دهند و مدل‌سازی کنند.

تعمیم‌پذیری: این مدل‌ها انعطاف‌پذیری و تعمیم‌پذیری بالاتری دارند و قادر به سازگاری با مجموعه داده‌ها و شرایط مختلف هستند که آن‌ها را برای پیش‌بینی توان بادی در سناریوهای مختلف مناسب می‌کند.

دقت بالاتر در پیش‌بینی: مدل‌های یادگیری ماشین مانند SVM، Random Forest و XGBoost دقت و کارایی بالاتری را نسبت به روش‌های آماری سنتی در پیش‌بینی توان بادی ارائه می‌دهند.

**۳. مدل‌های یادگیری عمیق مانند LSTM، GRU و CNN چگونه محدودیت‌های روش‌های یادگیری ماشین سنتی را در پیش‌بینی توان بادی برطرف می‌کنند؟**

یادگیری وابستگی‌های زمانی: مدل‌هایی مانند LSTM و GRU در یادگیری الگوهای زمانی و وابستگی‌های بلندمدت در داده‌های ترتیبی عملکرد بسیار خوبی دارند که برای پیش‌بینی توان بادی بسیار حیاتی است.

مدیریت ویژگی‌های پیچیده و غیرخطی: مدل‌های یادگیری عمیق برای استخراج و پردازش ویژگی‌های پیچیده و با ابعاد بالا در داده‌ها طراحی شده‌اند و می‌توانند از روش‌های سنتی یادگیری ماشین عملکرد بهتری داشته باشند.

مقیاس‌پذیری: این مدل‌ها برای مجموعه داده‌های بزرگ مناسب‌تر هستند زیرا به‌طور مستقیم از داده‌های خام یاد می‌گیرند و نیاز به مهندسی ویژگی‌های پیچیده ندارند.

**۴. اهمیت مکانیزم خودتوجهی (Self-Attention) در شبکه‌های Transformer برای یادگیری وابستگی‌های بلندمدت و همبستگی‌های محلی چیست؟**

مدل‌سازی وابستگی‌های جهانی: مکانیزم خودتوجهی در شبکه‌ی Transformer امکان مدل‌سازی وابستگی‌های جهانی در دنباله‌های طولانی را فراهم می‌کند و محدودیت‌های مدل‌های بازگشتی مانند LSTM را که فقط به‌صورت ترتیبی داده‌ها را پردازش می‌کنند، برطرف می‌کند.

محاسبات کارآمد: برخلاف مدل‌های مبتنی بر RNN، مکانیزم خودتوجهی پردازش موازی داده‌ها را ممکن می‌سازد، که موجب کاهش هزینه‌ی محاسباتی و بهبود مقیاس‌پذیری می‌شود.

تقویت درک همبستگی‌ها: توجه چندسری (Multi-Head Attention) در شبکه‌ی Transformer، امکان استخراج همبستگی‌های محلی و بلندمدت در داده‌های سری زمانی را فراهم کرده و دقت پیش‌بینی را افزایش می‌دهد.

**۵. تابع خطای Huber چگونه پایداری و استحکام مدل‌های پیش‌بینی توان بادی برای داده‌های فراساحلی را افزایش می‌دهد؟**

مدیریت داده‌های پرت: تابع خطای Huber بین خطای مربعی (برای خطاهای کوچک) و خطای قدرمطلق (برای خطاهای بزرگ) تغییر حالت می‌دهد، که آن را در برابر داده‌های پرت و تغییرات ناگهانی در داده‌های توان بادی مقاوم می‌کند.

بهبود بهینه‌سازی: پیوستگی و مشتق‌پذیری این تابع در سراسر دامنه، فرآیند بهینه‌سازی گرادیان نزولی را پایدارتر می‌کند و به همگرایی بهتر مدل کمک می‌کند.

افزایش استحکام: با کاهش تأثیر نوسانات و تصادفی بودن داده‌ها، تابع خطای Huber پایداری و قابلیت اطمینان مدل‌های پیش‌بینی توان بادی را افزایش می‌دهد.

۱-2. آماده‌سازی

۱. ساختار Autoencoder: برای چه هدفی استفاده می‌شود؟ از چه اجزایی تشکیل شده است؟

Autoencoder برای استخراج ویژگی‌ها و کاهش ابعاد داده‌های توان بادی استفاده می‌شود. هدف آن این است که داده‌های ورودی با ابعاد بالا را به یک نمایش فشرده با ابعاد پایین‌تر تبدیل کند، در حالی که اطلاعات مهم را حفظ می‌کند. این روش نویز داده‌ها را کاهش داده و کیفیت ویژگی‌ها را برای وظایف پیش‌بینی بهبود می‌بخشد.

اجزای آن عبارت‌اند از:

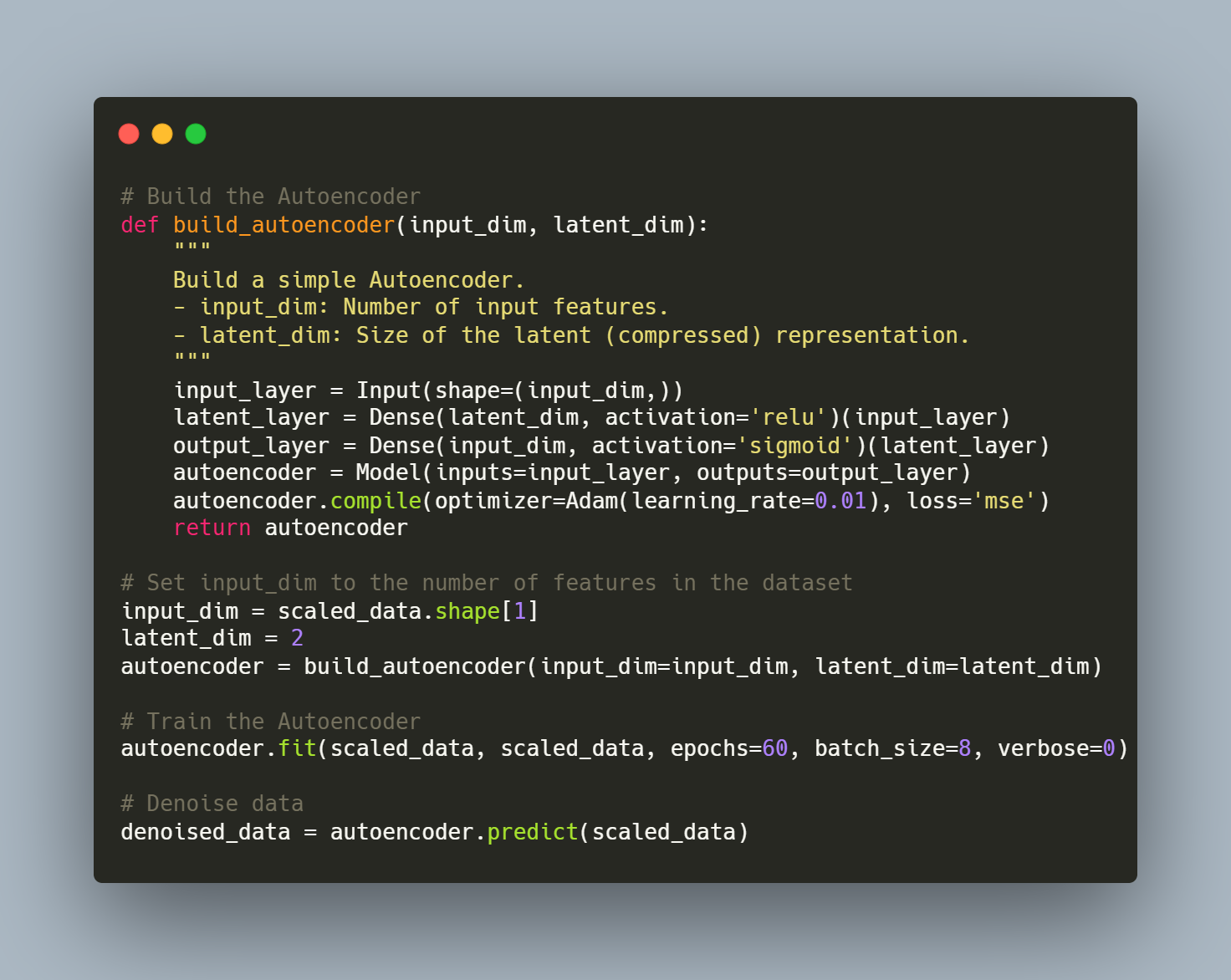
رمزگذار (Encoder): داده‌های ورودی را به یک نمایش فشرده و کوچک‌تر تبدیل می‌کند. این بخش شامل لایه‌های چگال (Dense) با توابع فعال‌سازی غیرخطی (مانند ReLU) برای استخراج ویژگی‌های معنادار است.

فضای نهان (Latent Space): نمایش فشرده داده‌های ورودی که ویژگی‌های اصلی را در خود جای داده است.

رمزگشا (Decoder): داده‌های ورودی را از فضای پنهان بازسازی می‌کند تا اطمینان حاصل شود که اطلاعات مهم حفظ شده است.

می‌توان Autoencoder را با استفاده از کتابخانه‌هایی مانند TensorFlow یا PyTorch پیاده‌سازی کرد. در این روش، encoder و decoder به عنوان شبکه‌های عصبی جداگانه تعریف شده و با کمینه کردن خطای بازسازی با یکدیگر آموزش داده می‌شوند.

در ادامه یک Autoencoder بسیار ساده و مطابق مقاله اشاره شده در سوال، پیاده‌سازی شده است:



**۲. مکانیزم توجه (Attention) و موقعیت‌یابی (Positional Encoding) در شبکه: برای چه هدفی استفاده می‌شود؟**

مکانیزم توجه: مکانیزم توجه به شبکه Transformer این امکان را می‌دهد که بر مهم‌ترین بخش‌های دنباله ورودی برای پیش‌بینی توان بادی متمرکز شود. این مکانیزم به مدل کمک می‌کند تا وابستگی‌های بلندمدت و روابط بین گام‌های زمانی را درک کند.

موقعیت‌یابی (Positional Encoding): از آنجا که Transformerها ساختار پردازش ترتیبی مانند RNNها ندارند، موقعیت‌یابی به مدل کمک می‌کند تا اطلاعات مربوط به ترتیب زمانی هر گام را به دنباله اضافه کند. این امر مدل را قادر می‌سازد که ترتیب داده‌ها را درک کند.

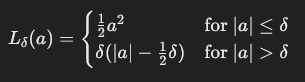
در ادامه به جزئیات می‌پردازیم:

مکانیزم توجه: مقاله از توجه چندسری (Multi-Head Attention) برای محاسبه سهم هر گام زمانی ورودی نسبت به دیگر گام‌ها استفاده می‌کند. این امر مدل را برای درک وابستگی‌های کوتاه و بلندمدت مؤثر می‌سازد.

موقعیت‌یابی: ترکیبی از توابع سینوسی و کسینوسی با فرکانس‌های مختلف برای اضافه کردن اطلاعات موقعیتی به داده‌های ورودی به کار گرفته می‌شود.

**۳. رابطه ریاضی تابع خطای Huber: رفتار این تابع چیست و هدف از استفاده آن در این آزمایش چیست؟**

فرمول ریاضی:



که در آن a=ytrue− pred​، و δ یک پارامتر آستانه است.

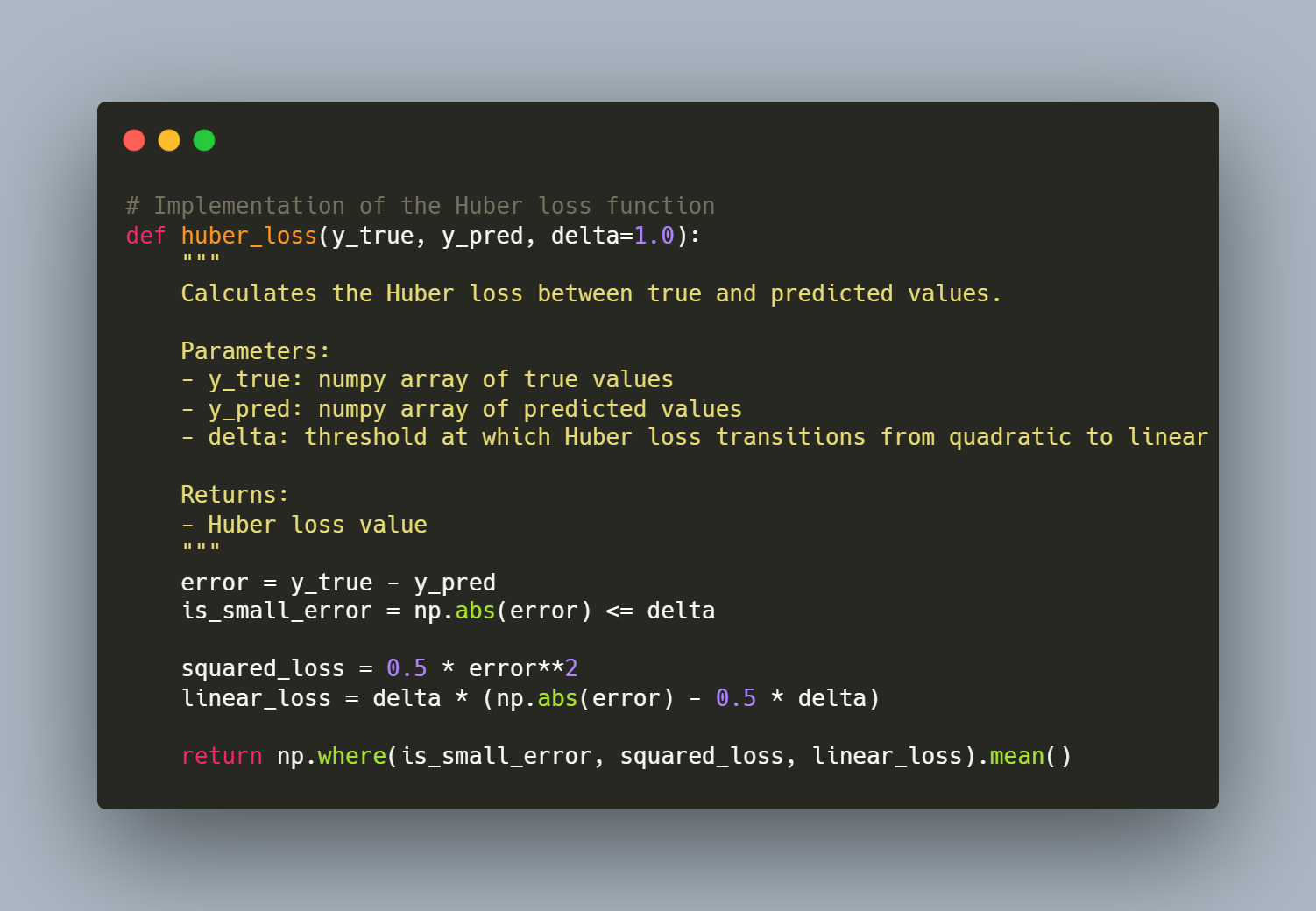
زمانی که خطا ∣a∣ کوچک باشد (کمتر از δ)، این تابع مانند خطای مربعی میانگین (MSE) رفتار کرده و جریمه‌ای مربعی اعمال می‌کند.

زمانی که خطا ∣a∣ بزرگ باشد (بیشتر از δ)، این تابع به خطای مطلق میانگین (MAE) تغییر حالت می‌دهد و تأثیر داده‌های پرت را کاهش می‌دهد.

مدیریت نویز و نوسانات: تابع Huber برای مدیریت طبیعت تصادفی و پرنوسان داده‌های توان بادی فراساحلی استفاده می‌شود.

پایداری و استحکام مدل: این تابع با کاهش تأثیر داده‌های پرت و تغییرات ناگهانی، پایداری و قابلیت اطمینان مدل را بهبود می‌بخشد.

در ادامه پیاده سازی آن در پایتون آمده است:



**۴. روش کار الگوریتم Slime Mould: آن را توضیح دهید و متغیرهای ورودی آن را پیاده‌سازی کنید.**

الهام‌گیری: الگوریتم Slime Mould یک الگوریتم بهینه‌سازی است که از رفتار نوسانی و حرکت تطبیقی موجودات کپک لجن‌مانند در جستجوی غذا الهام گرفته شده است.

هدف: این الگوریتم توازن بین جستجوی مناطق جدید (Exploration) و تمرکز بر مناطق امیدوارکننده (Exploitation) را برای یافتن بهترین راه‌حل حفظ می‌کند.

مراحل:

به‌روزرسانی وزن‌ها: وزن‌های کپک‌های لجن بر اساس تناسب آن‌ها تنظیم می‌شود که فرآیند غلظت غذا را شبیه‌سازی می‌کند.

حرکت تصادفی: کپک‌ها در فضای جستجو از طریق حرکت‌های تصادفی حرکت می‌کنند که تحت تأثیر موقعیت بهترین راه‌حل یافت‌شده قرار دارند.

کنترل مرزها: موقعیت کپک‌های لجن بررسی می‌شود تا مطمئن شویم که در محدوده فضای جستجو باقی می‌مانند.

متغیرهای ورودی:

اندازه جمعیت: تعداد عوامل کپک در الگوریتم.

حداکثر تکرار: حداکثر تعداد نسل‌ها برای فرآیند بهینه‌سازی.

تابع تناسب: تابع هدف برای کمینه یا بیشینه کردن (مانند MSE در این آزمایش).

محدوده فضای جستجو: محدوده مقادیر ممکن برای هر پارامتر بهینه‌سازی (مانند نرخ یادگیری، اندازه دسته، و غیره).

پیاده‌سازی:

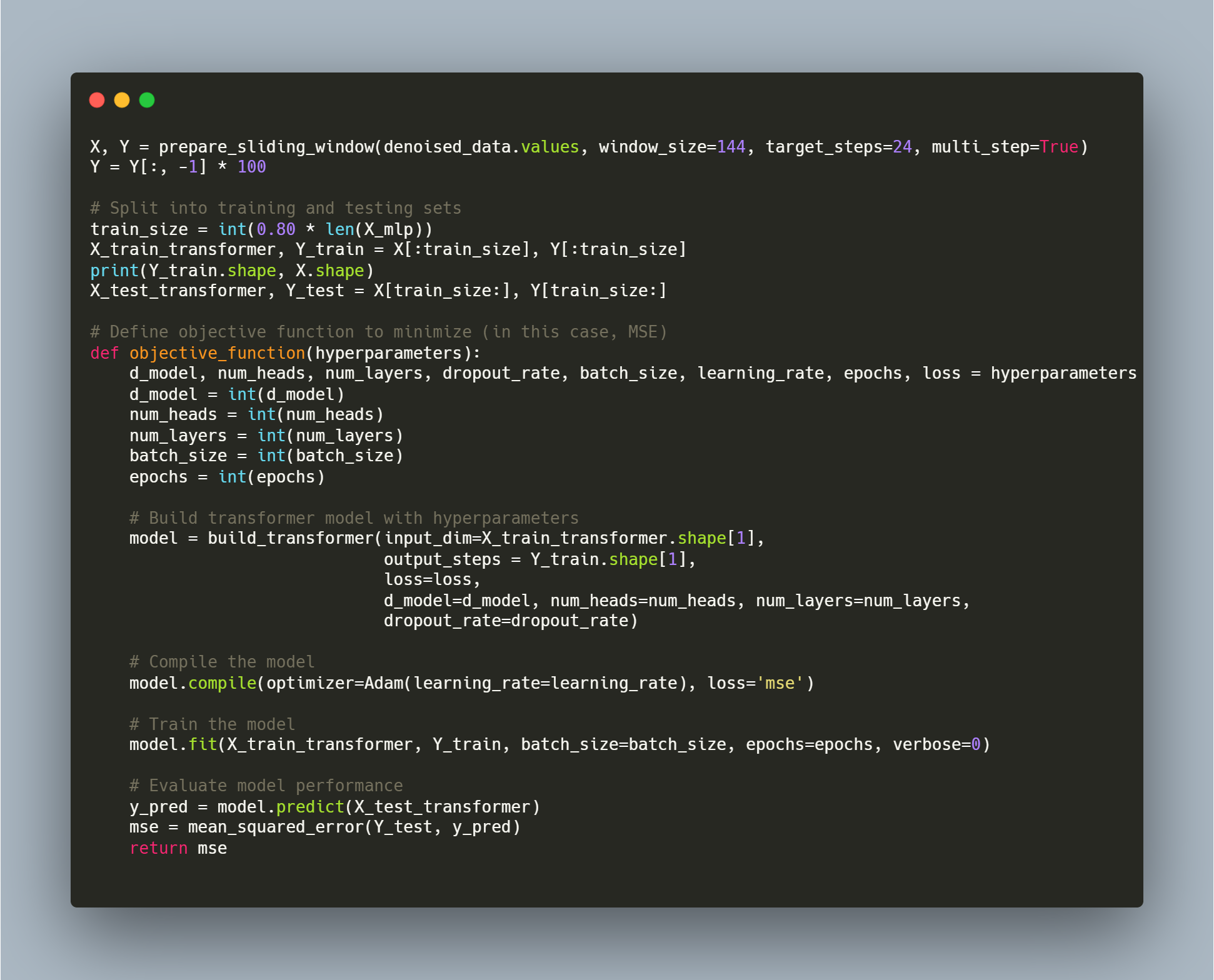
تعریف تابع هدف: تابع ارزیابی مدل (مانند MSE).

مقداردهی اولیه جمعیت: جمعیت کپک‌ها به‌صورت تصادفی در فضای جستجو مقداردهی اولیه می‌شود.

به‌روزرسانی موقعیت‌ها: موقعیت کپک‌ها به‌صورت تکراری بر اساس قوانین الگوریتم به‌روزرسانی می‌شود.

بازگشت بهترین راه‌حل: بهترین راه‌حل و مقدار تناسب آن پس از رسیدن به معیار توقف بازگردانده می‌شود.

پیاده‌سازی در پایتون:

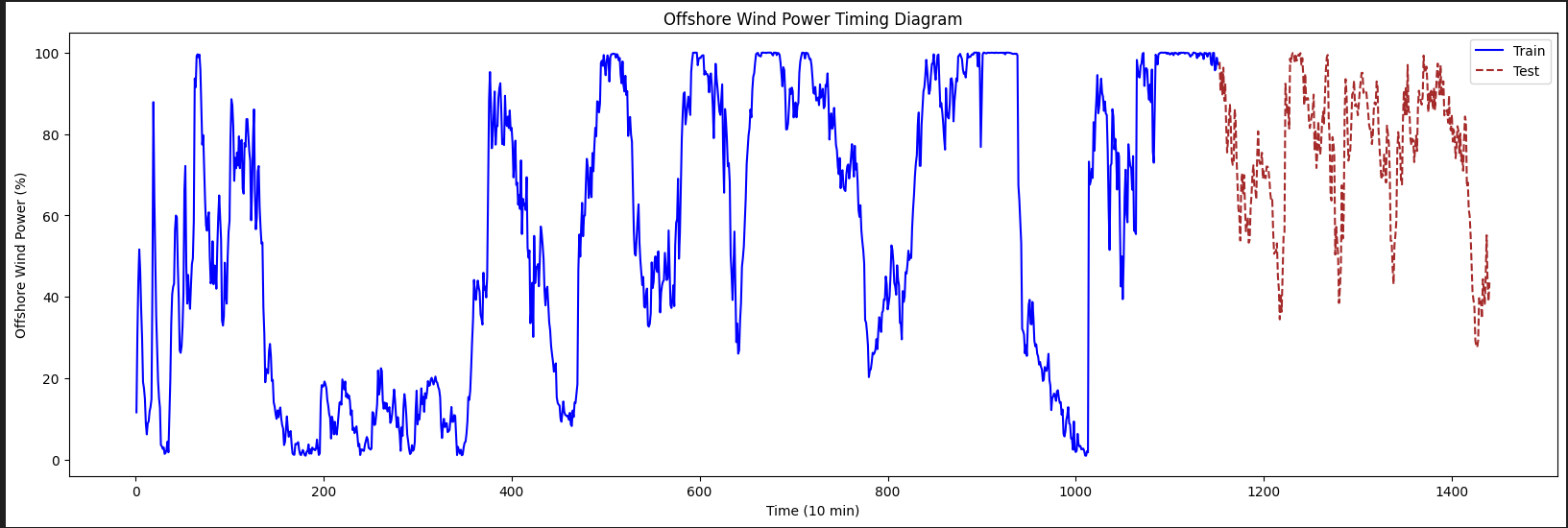


A screen shot of a computer screen

Description automatically generated

۱-3. روش‌شناسی و نتایج

۱.



2.

A graph showing a graph

Description automatically generated with medium confidence

3.

A graph showing a graph

Description automatically generated with medium confidence

4.



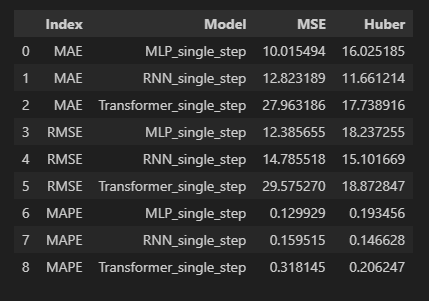
اگر نرمال‌سازی اعمال نشود، مدل ممکن است به دلیل تفاوت در مقیاس ویژگی‌های ورودی، به‌درستی همگرا نشود.  
اگر از روش پنجره لغزان استفاده نشود، وابستگی‌های زمانی موجود در داده‌ها شناسایی نخواهند شد، که منجر به پیش‌بینی‌های ضعیف می‌شود.  
اگر داده‌ها به مجموعه‌های آموزشی و آزمایشی تقسیم نشوند، مدل قادر به تعمیم‌دهی به داده‌های دیده‌نشده نخواهد بود.

5.

A screen shot of a computer program

Description automatically generated

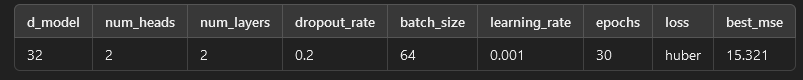
6.



7.

A screen shot of a computer screen

Description automatically generated



8. مرحله ۱: آماده‌سازی داده با روش پنجره لغزان

از تابع prepare\_sliding\_window برای ایجاد دنباله‌های ورودی (X) و مقادیر هدف (Y) استفاده شده است.

تنظیمات:

طول پنجره: ۱۴۴

تعداد گام‌های هدف: ۲۴

پیش‌بینی تک‌مرحله‌ای: multi\_step=False

مقدار هدف آخرین مقدار پنجره پیش‌بینی است:

X, Y = prepare\_sliding\_window(denoised\_data.values, window\_size=144, target\_steps=24, multi\_step=False)

مرحله ۲: تقسیم داده‌ها به مجموعه‌های آموزش و تست

مجموعه آموزش: ۸۰٪ داده‌ها برای آموزش مدل.

مجموعه تست: ۲۰٪ باقی‌مانده برای ارزیابی مدل.

train\_size = int(0.80 \* len(X))

X\_train, Y\_train = X[:train\_size], Y[:train\_size]

X\_test, Y\_test = X[train\_size:], Y[train\_size:]

مرحله ۳: آموزش مدل‌ها

مدل‌های استفاده‌شده:

MLP: شبکه عصبی چندلایه.

RNN: شبکه بازگشتی برای درک وابستگی‌های زمانی.

Transformer: استفاده از مکانیزم توجه برای مدل‌سازی دنباله‌ها.

تابع ارزیابی و آموزش: مدل‌ها با دو تابع هزینه MSE و Huber آموزش داده شدند و معیارهای MAE، RMSE و MAPE ارزیابی شدند.

train\_and\_evaluate(X\_train, Y\_train, X\_test, Y\_test, build\_model\_fn=build\_mlp, loss='mse', ...)

مرحله ۴: ذخیره و نمایش نتایج

نتایج ارزیابی شامل معیارها (MAE، RMSE، MAPE) در قالب یک جدول ذخیره و نمایش داده شدند:

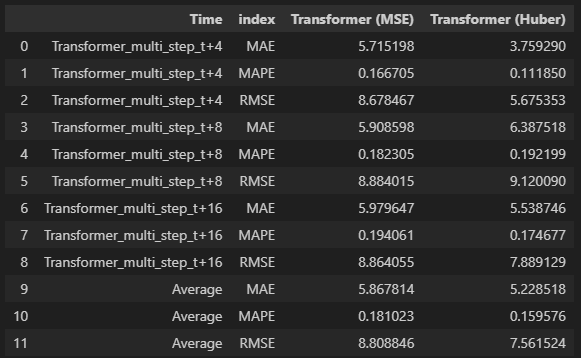
results\_table = results\_df.reset\_index()

هر گام زمانی در مجموعه داده‌ها نمایانگر یک بازه ۱۰ دقیقه‌ای است. بنابراین:

- t+4: ۴۰ دقیقه جلوتر

- t+8: ۸۰ دقیقه جلوتر

- t+16: ۱۶۰ دقیقه جلوتر



.

# **پرسش ۲** **-** استفاده از ViTبرای طبقهبندی تصاویر گلبولهای سفید

## ۱-۲.مقدمه

۲-۲ .آماده سازی داده ها

نمایش نمونه تصاویر:

import os

import matplotlib.pyplot as plt

from tensorflow.keras.preprocessing.image import load\_img, img\_to\_array

from collections import Counter

# Define paths

dataset\_path = "/content/drive/MyDrive/Q2\_Data"  # Replace with your BCCD dataset path

classes = ["NEUTROPHIL", "EOSINOPHIL", "LYMPHOCYTE", "MONOCYTE"]

# for cls in classes:

#     class\_path = os.path.join(dataset\_path, cls)

#     print(f"Contents of {class\_path}:")

#     print(os.listdir(class\_path) if os.path.exists(class\_path) else "Folder not found.")

# Display one sample image per class

def display\_sample\_images():

    plt.figure(figsize=(12, 8))

    for i, cls in enumerate(classes):

        class\_path = os.path.join(dataset\_path, cls)

        sample\_image = os.listdir(class\_path)[0]  # Get one image

        img = load\_img(os.path.join(class\_path, sample\_image))

        plt.subplot(1, len(classes), i + 1)

        plt.imshow(img)

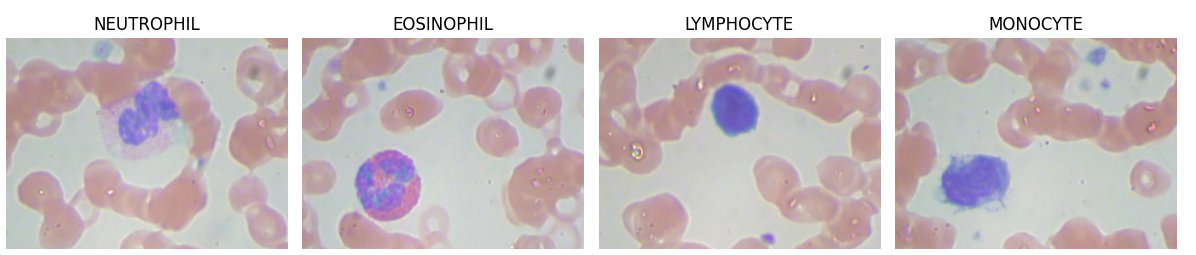
        plt.title(cls)

        plt.axis("off")

    plt.tight\_layout()

    plt.show()

display\_sample\_images()



بررسی تعداد دادهها در هر کلاس:

# Check class distribution

def check\_data\_distribution():

    class\_counts = {}

    for cls in classes:

        class\_path = os.path.join(dataset\_path, cls)

        class\_counts[cls] = len(os.listdir(class\_path))

    # Plot distribution

    plt.bar(class\_counts.keys(), class\_counts.values(), color='green')

    plt.xlabel('Classes')

    plt.ylabel('Number of Images')

    plt.title('Class Distribution')

    plt.show()

    # Check if dataset is imbalanced

    print("Class Counts:", class\_counts)

    max\_count = max(class\_counts.values())

    is\_imbalanced = any(count < 0.8 \* max\_count for count in class\_counts.values())

    if is\_imbalanced:

        print("The dataset is imbalanced.")

    else:

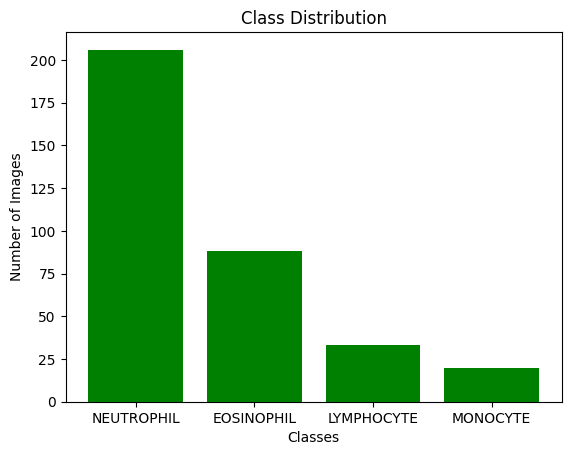
        print("The dataset is balanced.")

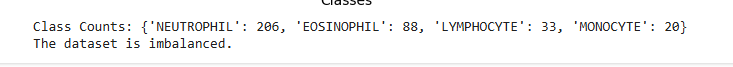
    return class\_counts

class\_counts = check\_data\_distribution()

This bar chart titled "Class Distribution" visualizes the number of images in different classes.

1. **Classes**: The x-axis represents four categories or classes: NEUTROPHIL, EOSINOPHIL, LYMPHOCYTE, and MONOCYTE. These might correspond to types of blood cells or categories in a classification dataset.
2. **Number of Images**: The y-axis indicates the count of images associated with each class.
3. **Observation**:
   * **NEUTROPHIL**: This class has the highest count, with approximately 200 images.
   * **EOSINOPHIL**: The second-highest, with about 100 images.
   * **LYMPHOCYTE**: Significantly fewer, around 25 images.
   * **MONOCYTE**: The smallest class, with slightly fewer than 25 images.
4. **Distribution**: The dataset is imbalanced, with a predominance of NEUTROPHIL images and a much smaller representation of LYMPHOCYTE and MONOCYTE images. This imbalance may affect training machine learning models, as models might favor the larger class unless adjustments (e.g., class weighting, oversampling, or undersampling) are made





from tensorflow.keras.preprocessing.image import ImageDataGenerator

import shutil

# Data augmentation

def augment\_data(target\_count):

    augmented\_path = "augmented\_dataset"

    if not os.path.exists(augmented\_path):

        os.makedirs(augmented\_path)

    for cls in classes:

        class\_path = os.path.join(dataset\_path, cls)

        augmented\_class\_path = os.path.join(augmented\_path, cls)

        if not os.path.exists(augmented\_class\_path):

            os.makedirs(augmented\_class\_path)

        images = os.listdir(class\_path)

        gen = ImageDataGenerator(

            rescale=1.0 / 255,  # Normalize pixel values

            rotation\_range=30,  # Random rotations

            zoom\_range=0.2,     # Random zoom

            shear\_range=0.2,    # Random shearing

            width\_shift\_range=0.2,  # Random horizontal shift

            height\_shift\_range=0.2,  # Random vertical shift

            horizontal\_flip=True,  # Random horizontal flip

            vertical\_flip=True,  # Random vertical flip

            brightness\_range=[0.8, 1.2],  # Random brightness adjustment

            fill\_mode='nearest'  # Fill pixels after transformation

        )

        # Augment until target\_count is reached

        for img\_name in images:

            img\_path = os.path.join(class\_path, img\_name)

            img = load\_img(img\_path, target\_size=(224, 224))

            x = img\_to\_array(img)

            x = x.reshape((1,) + x.shape)

            for batch in gen.flow(x, batch\_size=1, save\_to\_dir=augmented\_class\_path, save\_prefix=cls, save\_format='jpeg'):

                if len(os.listdir(augmented\_class\_path)) >= target\_count:

                    break

# Balance the dataset by augmenting

max\_samples = max(class\_counts.values())

for cls, count in class\_counts.items():

    if count < max\_samples:

        augment\_data(max\_samples)

print("Data augmentation completed.")



ایجاد مجموعه داده نهایی:

from sklearn.model\_selection import train\_test\_split

import shutil

import os

# Split dataset into training and validation

def split\_dataset():

    train\_path = "train\_dataset"

    val\_path = "val\_dataset"

    if not os.path.exists(train\_path):

        os.makedirs(train\_path)

    if not os.path.exists(val\_path):

        os.makedirs(val\_path)

    for cls in classes:

        augmented\_class\_path = os.path.join("augmented\_dataset", cls)

        images = os.listdir(augmented\_class\_path)

        # Ensure you have enough samples for splitting

        if len(images) > 1:

            train\_images, val\_images = train\_test\_split(images, test\_size=0.1, random\_state=42)

            train\_class\_path = os.path.join(train\_path, cls)

            val\_class\_path = os.path.join(val\_path, cls)

            os.makedirs(train\_class\_path, exist\_ok=True)

            os.makedirs(val\_class\_path, exist\_ok=True)

            for img\_name in train\_images:

                shutil.copy(os.path.join(augmented\_class\_path, img\_name), train\_class\_path)

            for img\_name in val\_images:

                shutil.copy(os.path.join(augmented\_class\_path, img\_name), val\_class\_path)

    print(f"Data split completed: {len(train\_images)} training, {len(val\_images)} validation samples per class.")

split\_dataset()



from tensorflow.keras.models import Model

from tensorflow.keras.layers import Dense, Dropout, BatchNormalization

from tensorflow.keras.optimizers import Adam

from tensorflow.keras.applications import EfficientNetV2B0

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.regularizers import l2

from sklearn.utils.class\_weight import compute\_class\_weight

import numpy as np

import tensorflow as tf

# Define paths

train\_path = "train\_dataset"

val\_path = "val\_dataset"

input\_shape = (224, 224, 3)

batch\_size = 16

# 32

num\_classes = 4

# Data generators for training and validation

train\_datagen = ImageDataGenerator(rescale=1.0 / 255)

val\_datagen = ImageDataGenerator(rescale=1.0 / 255)

train\_generator = train\_datagen.flow\_from\_directory(

    train\_path,

    target\_size=(224, 224),

    batch\_size=batch\_size,

    class\_mode="categorical",

)

val\_generator = val\_datagen.flow\_from\_directory(

    val\_path,

    target\_size=(224, 224),

    batch\_size=batch\_size,

    class\_mode="categorical",

)

# Compute class weights

class\_weights = compute\_class\_weight(

    class\_weight="balanced",

    classes=np.unique(train\_generator.classes),

    y=train\_generator.classes

)

class\_weights = dict(enumerate(class\_weights))

print("Class Weights:", class\_weights)

# Load Vision Transformer model "imagenet"

base\_model = EfficientNetV2B0(weights= "imagenet", include\_top=False, input\_shape=input\_shape, pooling="avg")

x = base\_model.output

x = Dense(512, activation="relu")(x)

x = Dropout(0.5)(x)  # Add dropout layer

x = BatchNormalization()(x)  # Add Batch Normalization

x = Dense(num\_classes, activation="softmax")(x)

# x = base\_model.output

# # Add fully connected layers with dropout, batch normalization, and L2 regularization

# x = Dense(512, kernel\_regularizer=l2(0.01))(x)

# x = BatchNormalization()(x)

# x = tf.keras.layers.Activation("relu")(x)

# x = Dropout(0.5)(x)  # Dropout to prevent overfitting

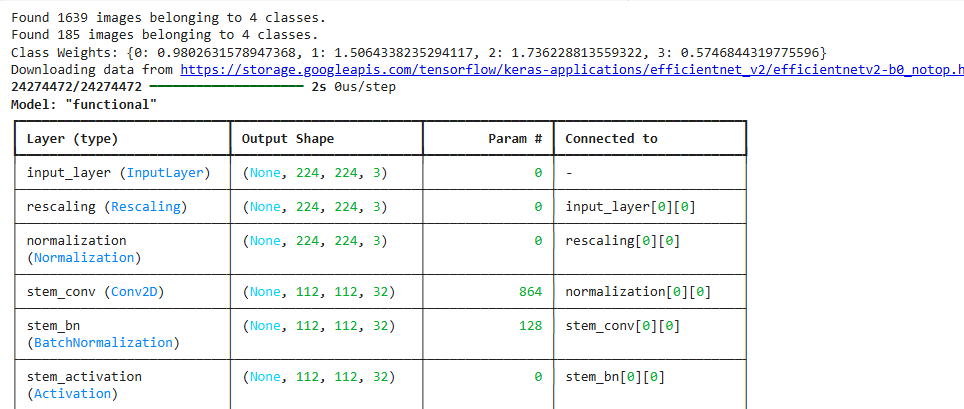
# x = Dense(num\_classes, activation="softmax")(x)

model = Model(inputs=base\_model.input, outputs=x)

model.summary()

# Compile the model

model.compile(optimizer=Adam(learning\_rate=1e-4), loss="categorical\_crossentropy", metrics=["accuracy"])



import numpy as np

def predict\_and\_visualize(mode\_name, model):

    print(f"\nVisualizing predictions for {mode\_name}...\n")

    # Reset the validation generator

    val\_generator.reset()

    images, labels = next(val\_generator)

    # Make predictions

    predictions = model.predict(images)

    predicted\_classes = np.argmax(predictions, axis=1)

    true\_classes = np.argmax(labels, axis=1)

    # Class labels

    class\_labels = list(val\_generator.class\_indices.keys())

    # Plot results

    plot\_images = (images \* 255).astype("uint8")

    plt.figure(figsize=(8, 6))

    for i in range(9):  # Display 9 images

        plt.subplot(3, 3, i + 1)

        plt.imshow(plot\_images[i])

        true\_label = class\_labels[true\_classes[i]]

        predicted\_label = class\_labels[predicted\_classes[i]]

        color = "green" if true\_classes[i] == predicted\_classes[i] else "red"

        plt.title(f"True: {true\_label}\nPred: {predicted\_label}", color=color)

        plt.axis("off")

    plt.tight\_layout()

    plt.show()

۳-۲ .آ آموزش مدلها

import matplotlib.pyplot as plt

from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau, ModelCheckpoint

# Early stopping, learning rate scheduler, and model checkpoint callbacks

early\_stopping = EarlyStopping(monitor="val\_loss", patience=5, restore\_best\_weights=True)

# Train the model

history = model.fit(

    train\_generator,

    validation\_data=val\_generator,

    epochs=50,  # Set epochs to a larger number, as early stopping will control the end

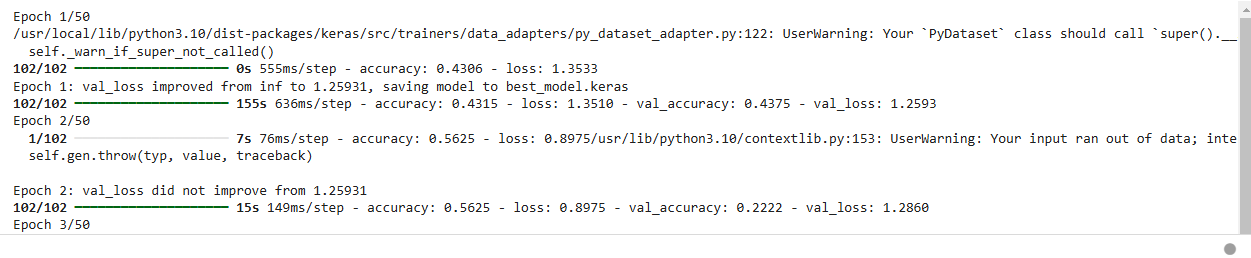
    steps\_per\_epoch=train\_generator.samples // batch\_size,

    validation\_steps=val\_generator.samples // batch\_size,

    class\_weight=class\_weights,

    callbacks=[early\_stopping],

)



# Plot accuracy and loss

def plot\_metrics(history):

    # Accuracy plot

    plt.figure(figsize=(12, 5))

    plt.subplot(1, 2, 1)

    plt.plot(history.history["accuracy"], label="Training Accuracy")

    plt.plot(history.history["val\_accuracy"], label="Validation Accuracy")

    plt.title("Accuracy")

    plt.xlabel("Epoch")

    plt.ylabel("Accuracy")

    plt.legend()

    # Loss plot

    plt.subplot(1, 2, 2)

    plt.plot(history.history["loss"], label="Training Loss")

    plt.plot(history.history["val\_loss"], label="Validation Loss")

    plt.title("Loss")

    plt.xlabel("Epoch")

    plt.ylabel("Loss")

    plt.legend()

    plt.tight\_layout()

    plt.show()

plot\_metrics(history)

These two plots represent the **training and validation metrics** during the training of a machine learning model, likely for a classification task.

### 1. ****Accuracy Plot (Left Panel)****:

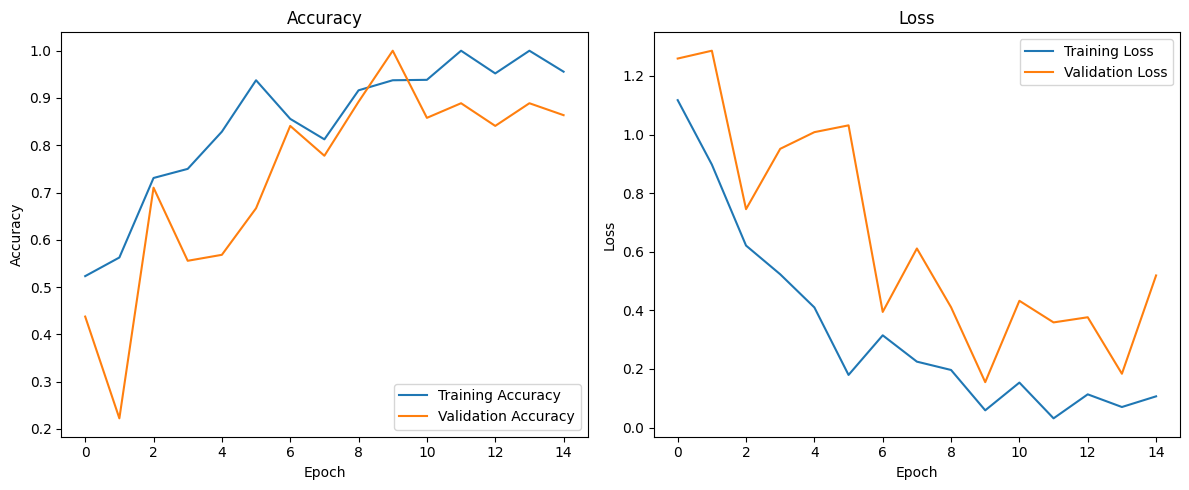
* **Purpose**: Tracks how the model's accuracy improves over epochs for both training and validation datasets.
* **Lines**:
  + **Training Accuracy (Blue Line)**: Represents how well the model performs on the training data.
  + **Validation Accuracy (Orange Line)**: Represents how well the model generalizes to unseen data (validation set).
* **Observations**:
  + Both training and validation accuracy increase over epochs, indicating that the model is learning effectively.
  + Around epoch 7, the validation accuracy peaks and closely matches the training accuracy, suggesting the model is generalizing well.
  + Slight fluctuations in the validation accuracy are normal and could be due to the dataset's inherent noise or variability.

### 2. ****Loss Plot (Right Panel)****:

* **Purpose**: Tracks the loss (error) during training for both the training and validation datasets.
* **Lines**:
  + **Training Loss (Blue Line)**: Shows the decrease in error on the training data.
  + **Validation Loss (Orange Line)**: Indicates how well the model generalizes to the validation set.
* **Observations**:
  + Training loss steadily decreases, showing that the model is minimizing the error on the training set.
  + Validation loss initially decreases but begins to fluctuate significantly after epoch 7. This might suggest:
    - Overfitting: The model may start to fit the training data too closely, losing generalization.
    - Noise in the validation data: The fluctuations might be due to random variations in the validation set.

### Key Insights:

* The model performs well initially, with training and validation accuracy closely aligned.
* However, the fluctuations in validation loss after epoch 7 may indicate the onset of overfitting.
* To address overfitting, techniques like early stopping, regularization, or dropout could be employed.



آموزش مدل ViTدر حالتهای مختلف

حالت :۱فقط دستهبند Classifierقابل آموزش باشد

# Helper function to freeze/unfreeze layers

def set\_trainable\_layers(model, trainable\_layers):

    for layer in model.layers:

        layer.trainable = False  # Freeze all layers by default

    for layer\_name in trainable\_layers:

        model.get\_layer(layer\_name).trainable = True

    print(f"Trainable layers: {[layer.name for layer in model.layers if layer.trainable]}")

# Calculate parameters

def calculate\_parameters(model):

    total\_params = model.count\_params()

    trainable\_params = sum([K.count\_params(w) for w in model.trainable\_weights])

    non\_trainable\_params = total\_params - trainable\_params

    return total\_params, trainable\_params, non\_trainable\_params

)Fine-tune First 2 Layers( Encoder : آموزش دو لایه اول

from tensorflow.keras import backend as K

# Mode 1: Freeze all layers except the classifier

set\_trainable\_layers(model, ["dense", "dense\_1"]) # Assuming dense layers are classifier layers

total\_params, trainable\_params, non\_trainable\_params = calculate\_parameters(model)

print(f"Total parameters: {total\_params}")

print(f"Trainable parameters: {trainable\_params}")

print(f"Non-trainable parameters: {non\_trainable\_params}")

# Compile and train the model

model.compile(optimizer=Adam(learning\_rate=1e-4), loss="categorical\_crossentropy", metrics=["accuracy"])

epochs = 50

history\_mode1 = model.fit(

    train\_generator,

    validation\_data=val\_generator,

    epochs=epochs,

    steps\_per\_epoch=train\_generator.samples // batch\_size,

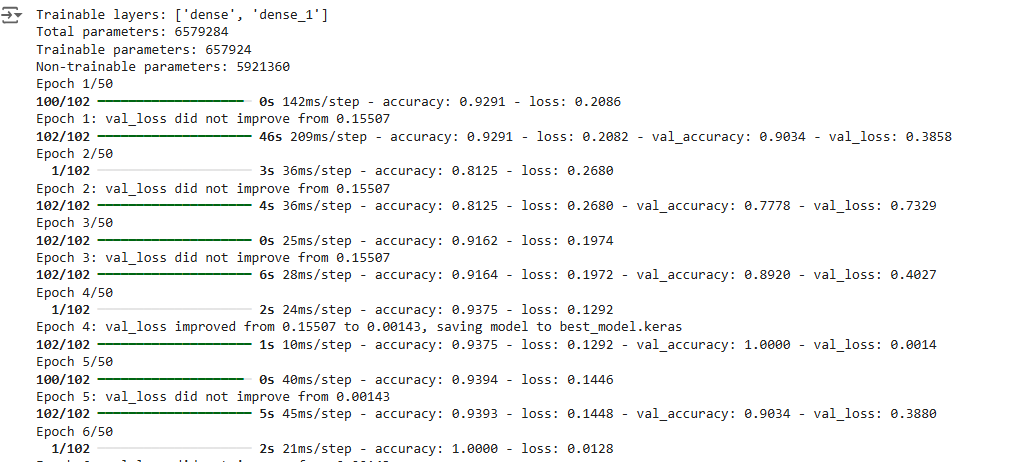
    validation\_steps=val\_generator.samples // batch\_size,

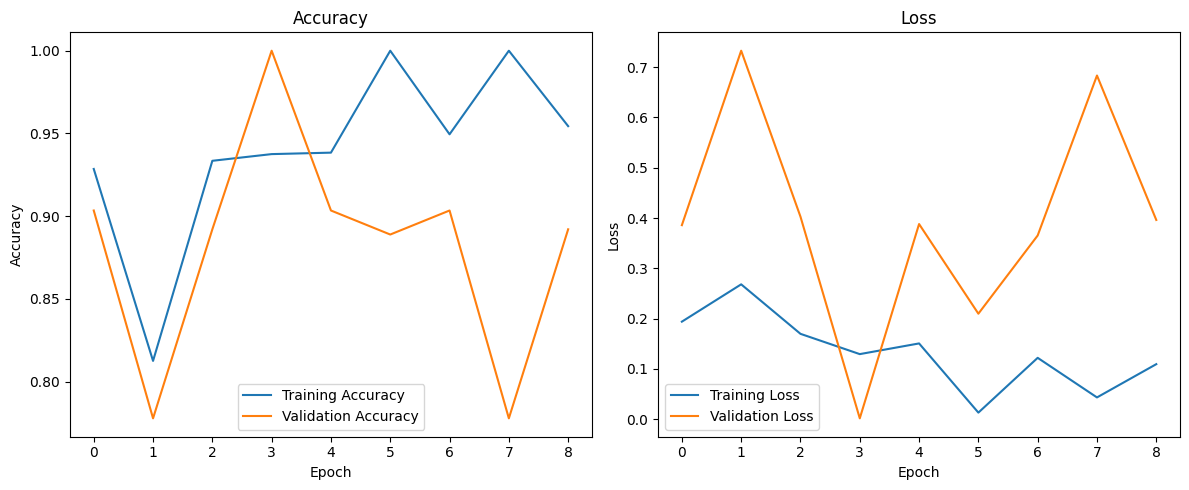
    class\_weight=class\_weights,

    callbacks=[early\_stopping],

)

plot\_metrics(history\_mode1)





These plots show the **accuracy** and **loss** metrics for a machine learning model's training and validation over 8 epochs.

### 1. ****Accuracy Plot (Left Panel)****:

* **Training Accuracy (Blue Line)**:
  + Gradually increases and stabilizes near 1.0, indicating that the model fits the training data very well.
* **Validation Accuracy (Orange Line)**:
  + Fluctuates significantly across epochs. It reaches high values (close to 1.0) in some epochs but drops sharply in others.
* **Observations**:
  + The fluctuations in validation accuracy suggest variability in model performance on the validation set, likely due to:
    - Small validation dataset size.
    - High sensitivity to noise or specific data instances.
  + The training accuracy is consistently high, showing that the model learns well on the training set.

### 2. ****Loss Plot (Right Panel)****:

* **Training Loss (Blue Line)**:
  + Steadily decreases and stabilizes near a very low value, confirming that the model minimizes the error on the training set effectively.
* **Validation Loss (Orange Line)**:
  + Highly variable, with sharp peaks and troughs across epochs.
* **Observations**:
  + The instability in validation loss suggests overfitting, where the model is memorizing the training data but failing to generalize to unseen data.
  + The sharp spikes in validation loss correspond to the drops in validation accuracy.

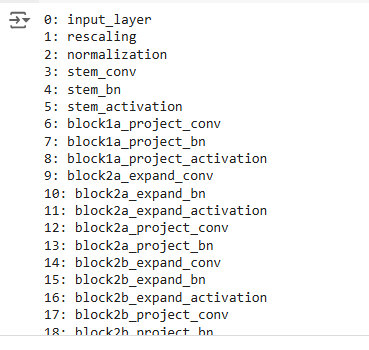
### Key Takeaways:

1. **Overfitting**: The model fits the training data very well but struggles to generalize to the validation set, as seen in the fluctuating validation metrics.
2. **Validation Set Issues**: The variability might indicate that the validation dataset is either small or not representative of the overall data distribution.
3. **Potential Solutions**:
   * Use regularization techniques like dropout or weight decay.
   * Perform data augmentation to increase diversity in training data.
   * Consider increasing the size of the validation set to reduce variability.
   * Employ early stopping to avoid training the model beyond the point where it starts overfitting

آموزش دو لایه **اول** (Fine-tune first two layers\_Encoder)

for i, layer in enumerate(model.layers):

    print(f"{i}: {layer.name}")



# Mode 2: Fine-tune the first two encoder layers and the classifier

set\_trainable\_layers(model, ["block1a\_project\_conv", "block2a\_expand\_conv", "dense", "dense\_1"])

print(f"Mode 2 - Total Parameters: {model.count\_params()}, Trainable: {len(model.trainable\_weights)}")

total\_params, trainable\_params, non\_trainable\_params = calculate\_parameters(model)

print(f"Total parameters: {total\_params}")

print(f"Trainable parameters: {trainable\_params}")

print(f"Non-trainable parameters: {non\_trainable\_params}")

# Compile and train the model

model.compile(optimizer=Adam(learning\_rate=1e-5), loss="categorical\_crossentropy", metrics=["accuracy"])

history\_mode2 = model.fit(

    train\_generator,

    validation\_data=val\_generator,

    epochs=epochs,

    steps\_per\_epoch=train\_generator.samples // batch\_size,

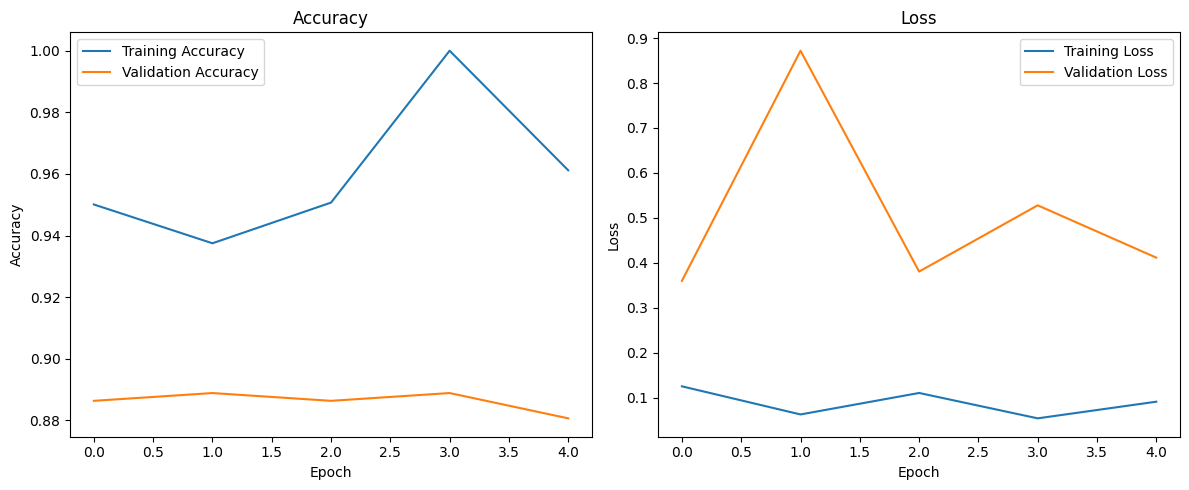
    validation\_steps=val\_generator.samples // batch\_size,

    class\_weight=class\_weights,

    callbacks=[early\_stopping],

)

plot\_metrics(history\_mode2)



These plots represent the training and validation performance of a machine learning model over a series of training epochs.

**Observations:**

1. The training accuracy improves over epochs, reaching a peak at epoch 3, and then slightly decreases at epoch 4. This indicates the model fits well on the training data initially but might overfit or encounter some irregularities.
2. Validation accuracy remains almost flat and much lower than training accuracy, suggesting the model struggles to generalize and might be overfitting to the training data.

**Observations:**

1. Training loss decreases consistently over epochs, showing the model is learning on the training data.
2. Validation loss spikes significantly at epoch 1 and then drops sharply, but remains higher than training loss throughout. This behavior suggests the model might be overfitting or is poorly calibrated for validation data.

### ****Key Insights****

* **Overfitting**: The gap between training accuracy and validation accuracy, as well as the disparity between validation loss and training loss, indicates overfitting.
* **Generalization Issue**: The validation metrics do not improve significantly, suggesting the model needs better regularization (e.g., dropout, L2 regularization) or more representative validation data.
* **Further Steps**: Consider techniques like cross-validation, reducing model complexity, or increasing the size/quality of the training dataset to improve generalization.

آموزش دو لایه آخر (Fine-tune last two layers\_Encoder)

# Mode 3: Fine-tune the last two encoder layers and the classifier

set\_trainable\_layers(model, ["block6h\_expand\_conv", "block6g\_expand\_conv", "dense", "dense\_1"])

print(f"Mode 3 - Total Parameters: {model.count\_params()}, Trainable: {len(model.trainable\_weights)}")

total\_params, trainable\_params, non\_trainable\_params = calculate\_parameters(model)

print(f"Total parameters: {total\_params}")

print(f"Trainable parameters: {trainable\_params}")

print(f"Non-trainable parameters: {non\_trainable\_params}")

# Compile and train the model

model.compile(optimizer=Adam(learning\_rate=1e-5), loss="categorical\_crossentropy", metrics=["accuracy"])

epochs = 50

history\_mode3 = model.fit(

    train\_generator,

    validation\_data=val\_generator,

    epochs=epochs,

    steps\_per\_epoch=train\_generator.samples // batch\_size,

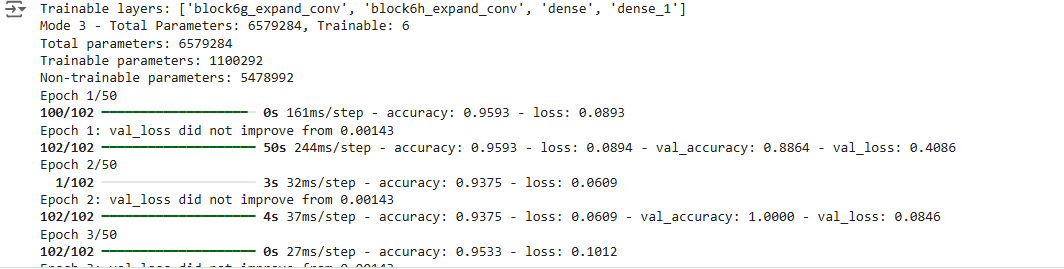
    validation\_steps=val\_generator.samples // batch\_size,

    class\_weight=class\_weights,

    callbacks=[early\_stopping],

)

plot\_metrics(history\_mode3)





These plots show the training and validation performance of a machine learning model in terms of accuracy and loss across epochs.

#### **Observations**:

1. The training accuracy starts high, dips slightly in the middle epochs, peaks at epoch 3, and then decreases slightly again.
2. The validation accuracy fluctuates wildly between epochs, reaching very high values (close to 1.0) at epoch 1 and epoch 3, but dropping significantly (below 0.9) at other epochs.

#### **Analysis**:

* The significant fluctuations in validation accuracy suggest instability in the model's generalization performance. This could be caused by:
  + **Small validation dataset**: The metrics may not represent the overall performance.
  + **Overfitting or high variance**: The model is highly sensitive to the specific validation data it encounters.
* The training accuracy is relatively stable, which indicates the model performs consistently on the training data.

#### **Observations**:

1. Training loss consistently decreases across epochs, indicating the model is learning and improving its fit to the training data.
2. Validation loss fluctuates significantly:
   * It is high at epochs 0, 2, and 4.
   * It drops sharply at epoch 3, aligning with the high validation accuracy at that epoch.

#### **Analysis**:

* These fluctuations in validation loss suggest overfitting or a lack of stability in the model's learning process.
* While the training loss follows a smooth and expected downward trend, the validation loss spikes and dips unpredictably, indicating that the model may not generalize well.

### ****Key Insights****:

1. **Validation Instability**: The sharp changes in validation accuracy and loss suggest that either the validation dataset is too small or the model is not generalizing well.
2. **Overfitting or High Variance**: The large disparity between training and validation metrics at various points suggests the model may be overfitting or is sensitive to noise in the data.

آموزش تمام لایه ها

# Mode 4: Fine-tune all layers

for layer in model.layers:

    layer.trainable = True

print(f"Mode 4 - Total Parameters: {model.count\_params()}, Trainable: {len(model.trainable\_weights)}")

total\_params, trainable\_params, non\_trainable\_params = calculate\_parameters(model)

print(f"Total parameters: {total\_params}")

print(f"Trainable parameters: {trainable\_params}")

print(f"Non-trainable parameters: {non\_trainable\_params}")

# Compile and train the model

model.compile(optimizer=Adam(learning\_rate=1e-6), loss="categorical\_crossentropy", metrics=["accuracy"])

epochs = 50

history\_mode4 = model.fit(

    train\_generator,

    validation\_data=val\_generator,

    epochs=epochs,

    steps\_per\_epoch=train\_generator.samples // batch\_size,

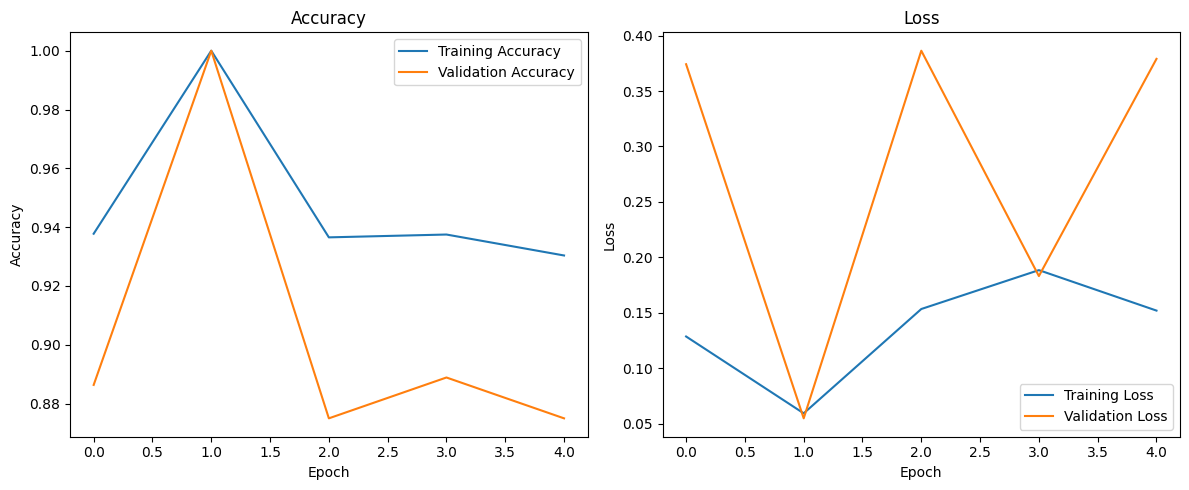
    validation\_steps=val\_generator.samples // batch\_size,

    class\_weight=class\_weights,

    callbacks=[early\_stopping],

)

plot\_metrics(history\_mode4)



These plots depict the performance of a machine learning model across training epochs, showing both accuracy and loss for training and validation datasets.

1. **Training Accuracy**:
   * Starts at a relatively high value (~0.94).
   * Peaks at epoch 1 (near 1.0), then stabilizes and decreases slightly across the subsequent epochs.
2. **Validation Accuracy**:
   * Starts low (~0.88), spikes to a very high value at epoch 1 (matching the training accuracy), but drops significantly afterward.
   * Remains consistently lower than the training accuracy after epoch 1.

#### **Analysis**:

* The large peak in validation accuracy at epoch 1 might indicate that the model overfit to the validation set temporarily or coincidentally performed well on specific validation data during that epoch.
* The gap between training and validation accuracy in later epochs suggests that the model struggles to generalize and is overfitting the training data.

1. **Training Loss**:
   * Decreases steadily, showing the model is learning from the training data as expected.
2. **Validation Loss**:
   * Drops sharply at epoch 1, corresponding to the peak in validation accuracy.
   * After epoch 1, it fluctuates dramatically, increasing significantly by epoch 2 and peaking again at epoch 4.

#### **Analysis**:

* The fluctuations in validation loss, combined with its high values in later epochs, indicate that the model's performance on validation data is unstable.
* The disparity between training and validation loss further supports the conclusion that the model is overfitting to the training data.

### ****Key Insights****:

1. **Overfitting**:
   * Training accuracy remains consistently high, while validation accuracy and loss fluctuate significantly, suggesting overfitting.
2. **Validation Instability**:
   * The large spike in validation accuracy at epoch 1, followed by a drop, highlights instability in generalization.
3. **Possible Issues**:
   * The validation dataset may be too small or not representative.
   * The model may lack regularization, leading to overfitting.

آموزش مدل :

from tensorflow.keras.applications import DenseNet121

from tensorflow.keras.layers import GlobalAveragePooling2D, Dropout

from tensorflow.keras.models import Model

from tensorflow.keras.optimizers import Adam

from tensorflow.keras.layers import Dense

from tensorflow.keras.regularizers import l2

# Load DenseNet-121 base model

base\_model = DenseNet121(weights="imagenet", include\_top=False, input\_shape=input\_shape)

x = base\_model.output

x = GlobalAveragePooling2D()(x)  # Global Average Pooling

# Add Dense layers with regularization and dropout

# x = Dense(512, activation="relu", kernel\_regularizer=l2(0.01))(x)

x = Dense(512, activation="relu")(x)

x = Dropout(0.5)(x)

x = Dense(num\_classes, activation="softmax")(x)

cnn\_model = Model(inputs=base\_model.input, outputs=x)

cnn\_model.summary()

# Fine-tune the model: Unfreeze all layers

for layer in base\_model.layers:

    layer.trainable = True

# Compile the model

cnn\_model.compile(optimizer=Adam(learning\_rate=1e-4), loss="categorical\_crossentropy", metrics=["accuracy"])

# Train the CNN model

epochs = 50

cnn\_history = cnn\_model.fit(

    train\_generator,

    validation\_data=val\_generator,

    epochs=epochs,

    steps\_per\_epoch=train\_generator.samples // batch\_size,

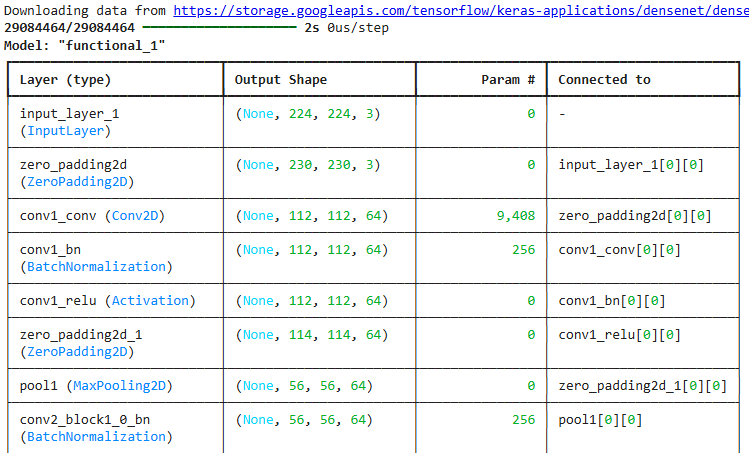
    validation\_steps=val\_generator.samples // batch\_size,

    class\_weight=class\_weights,

    callbacks=[early\_stopping],

)

plot\_metrics(cnn\_history)



total\_params = cnn\_model.count\_params()

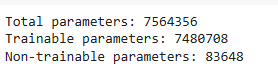
trainable\_params = sum([K.count\_params(w) for w in cnn\_model.trainable\_weights])

non\_trainable\_params = total\_params - trainable\_params

print(f"Total parameters: {total\_params}")

print(f"Trainable parameters: {trainable\_params}")

print(f"Non-trainable parameters: {non\_trainable\_params}")



# Freeze all layers except the classifier

for layer in base\_model.layers:

    layer.trainable = False  # Freeze all layers

# Only train classifier layers

cnn\_model.layers[-1].trainable = True

cnn\_model.layers[-2].trainable = True

cnn\_model.compile(optimizer=Adam(learning\_rate=1e-4), loss="categorical\_crossentropy", metrics=["accuracy"])

epochs = 50

# Train the classifier-only model

classifier\_history = cnn\_model.fit(

    train\_generator,

    validation\_data=val\_generator,

    epochs=epochs,

    steps\_per\_epoch=train\_generator.samples // batch\_size,

    validation\_steps=val\_generator.samples // batch\_size,

    class\_weight=class\_weights,

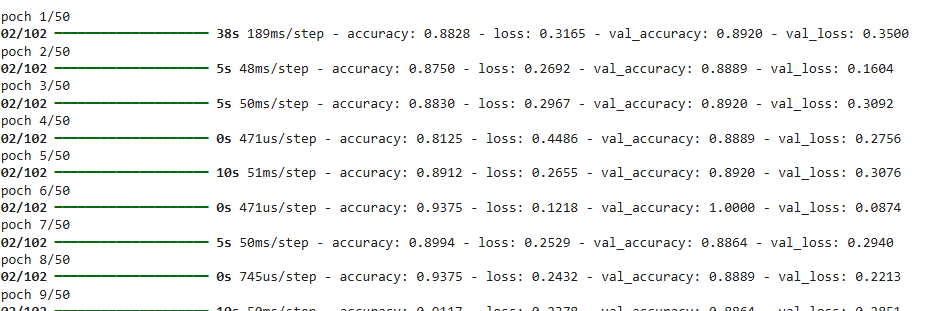
)

plot\_metrics(classifier\_history)

# Evaluate and compare with ViT results

print("Validation Accuracy (CNN Classifier Only):", max(classifier\_history.history["val\_accuracy"]))

print("Validation Loss (CNN Classifier Only):", min(classifier\_history.history["val\_loss"]))



۳-۲ تحلیل و نتیجه گیری

### ****Comparison of CNN and ViT Performance****

To compare the **CNN** and **ViT** models, their results should be evaluated based on metrics like **accuracy**, **loss**, **stability**, and **generalization to new data**.

**Which model performed better on your dataset?**

* When examining **accuracy** and **loss**:
  + **CNN**:
    - The training accuracy is high, but the gap between training and validation accuracy indicates **overfitting**.
    - Validation performance is relatively stable but doesn’t reach a very high accuracy level.
  + **ViT**:
    - The validation accuracy and loss show significant **fluctuations**, which could be due to the model's **dependency on large datasets** or its instability with smaller datasets.
    - While ViT achieves high accuracy at specific points, its performance is inconsistent, especially on validation data.

**Has ViT, under the current conditions (e.g., noisy or small datasets), proven to be a suitable replacement for CNN?**

* **ViT (Vision Transformer)** models typically require **large and diverse datasets** to perform well. They also need **careful tuning** and a substantial amount of data to stabilize learning.
* In the current scenario, where the dataset is small or noisy, **ViT struggles to generalize** and exhibits significant performance instability.
* **CNN** shows better performance due to its simpler architecture and stronger ability to extract local features, making it more resilient in these conditions.

### ****Summary:****

1. **Better Performance:** The **CNN** model outperformed ViT overall.
2. **ViT as a Replacement:** Under the current conditions, **ViT did not prove to be a viable alternative to CNN**. With a larger dataset, ViT might perform better, but for smaller or noisier datasets, **CNN is the more reliable option**.
3. The **CNN** outperforms ViT overall due to its relative stability and consistent performance on the validation set, even though both models have their limitations. **Conclusion:**  
   ViT is **not a suitable replacement for CNN** in situations involving **small or noisy datasets**. CNN remains the better choice due to its stability and adaptability in low-data scenarios.