دانشگاه تهران
دانشکده مهندسی برق و کامپیوتر

درس شبکه‌های عصبی و یادگیری عمیق
تمرین پنجم

مبینا مهرآذر - 810100216

محمدرضا محمدهاشمی - 810100206

فهرست

پرسش 1. پیش‌بینی نیروی باد به کمک مبدل و تابع خطای Huber 1

1-1. مقدمه 1

1-1-1. روش‌های آماری سنتی 1

1-1-2. مدل‌های یادگیری ماشین 1

1-1-3. مدل‌های یادگیری عمیق 2

1-1-4. مکانیزم خودتوجهی در شبکه مبدل 3

1-1-5. تابع خطای Huber 4

1-2. آماده‌سازی 5

1-2-1. ساختار Autoencoder 5

1-2-2. مکانیزم توجه 6

1-2-3. رمزگذاری موقعیت 6

1-2-4. رفتار تابع خطای Huber 7

1-2-5. الگوریتم Slime mould 7

نحوه عملکرد الگوریتم: 7

پارامترهای ورودی: 8

روند اجرای الگوریتم: 8

1-3. روش‌شناسی و نتایج 9

1-3-1. در حالت Single Step 12

1-3-3. در حالت Multi Step با t+4 13

1-3-4. در حالت Multi Step با t+8 14

1-3-4. در حالت Multi Step با t+16 14

1-3-5. مقایسه مدل‌های Multi Step 15

1-3-6. مقایسه مدل‌های مختلف با معماری‌ها منحصر به فرد 16

پرسش 2 - استفاده از ViT برای طبقه‌بندی تصاویر گلبول‌های سفید 19

2-1. مقدمه 19

2-2. آماده‌سازی داده‌ها 19

2-2-1. مشاهده تصاویر مجموعه داده 19

2-2-2. بررسی توزیع کلاس‌های مجموعه داده 19

علت استفاده از updampling، این است که تعداد داده کافی در اختیار مدل‌ها قرار دهیم و همینطور، نیاز است تعداد داده‌ها برابری کنند تا مدل‌ها با تعداد داده‌ی مساوی به خوبی آموزش ببینند. 20

2-2-3. داده‌افزایی 20
2-2-4. تقسیم داده 21
2-3. آموزش مدل‌ها 21
2-3-1. توضیح مدل ViT 22
الف. تقسیم تصویر به پچ‌ها ((Patch Embedding 22
ب. توکن‌های موقعیت ((Positional Embeddings 23
ج. لایه‌های ترنسفورمر 23
د. لایه‌های خروجی ((Head 23
مزایای Vision Transformer 23
معایب Vision Transformer 23
2-3-2. آموزش مدل ViT با حالت اول 24
2-3-3. بررسی نتایج مدل ViT با حالت اول 25
2-3-4. آموزش مدل ViT با حالت دوم 25
2-3-5. بررسی نتایج مدل ViT با حالت دوم 26
2-3-6. آموزش مدل ViT با حالت سوم 26
2-3-7. بررسی نتایج مدل ViT با حالت سوم 27
2-3-8. آموزش مدل ViT با حالت چهارم 27
2-3-9. بررسی نتایج مدل ViT با حالت چهارم 28
2-3-10. آموزش مدل CNN به طور کامل آموزش دیده 28
2-3-11. بررسی نتایج مدل CNN به طور کامل آموزش دیده 29
2-3-12. آموزش مدل CNN با فقط لایه‌ی Classifier آموزش دیده 29
2-3-13. بررسی نتایج مدل CNN با فقط لایه‌ی Classifier آموزش دیده 30
2-4. تحلیل و نتیجه‌گیری 30
2-4-1. مقایسه مدل‌های آموزش دیده روی تمامی لایه‌ها 30
2-4-2. مقایسه مدل‌های آموزش دیده روی لایه Classifier 32
2-4-3. مقایسه مدل‌های ViT در حالت‌های مختلف 33

پرسش 1. پیش‌بینی نیروی باد به کمک مبدل و تابع خطای Huber

1-1. مقدمه

1-1-1. روش‌های آماری سنتی

روش‌های آماری سنتی ماند میانگین متحرک یکپارچه خودرگرسیون (ARIMA) و ناهمسانی شرطی خودرگرسیون تعمیم یافته (GARCH)، دارای محدودیت‌هایی هستند:

این روش‌ها به فرضیات توزیعی دقیق[1] و آزمون‌های هموواری روی داده‌ها نیاز دارند. این مورد تعمیم‌پذیری و سازگاری آنها را با زمینه‌ها و شرایط مختلف محدود می‌کند.

این روش‌ها برای مدیریت موثر روابط پیچیده و غیرخطی و وابستگی‌های بلندمدت که اغلب در دنیای واقعی اینگونه است، به طور ایده‌آل کاربرد ندارند. در مقابل، مدل‌های یادگیری عمیق مانند شبکه‌های عصبی حافظه کوتاه‌مدت (LSTM)، شبکه‌های عصبی بازگشتی واحد دردار (GRU) و شبکه‌های عصبی کانولوشنال (CNN) برای این کار مناسب‌تر هستند. آنها می توانند الگوهای پیچیده را بیاموزند، غیرخطی بودن را پیمایش کنند و وابستگی های بلندمدت در داده ها را مدیریت کنند و در نتیجه پیش بینی کوتاه مدت انرژی باد را بهبود ببخشند.

1-1-2. مدل‌های یادگیری ماشین

مدل‌های یادگیری ماشین مانند SVM، جنگل تصادفی[2]، و XGboost مزیت‌های زیر را دارند:

قابلیت‌های پردازش داده قوی: این مدل‌ها می‌توانند به طور موثر مجموعه داده‌های بزرگ و پیچیده را مدیریت کرده و آنها را برای پیش‌بینی مسائلی که اغلب شامل چندین متغیر است، مناسب می‌سازد.

دقت پیش‌بینی بهبود یافته: این مدل‌ها در مقایسه با روش‌های آماری سنتی اغلب پیش‌بینی‌های بهتری ارائه می‌دهند.

تعمیم پذیری پیشرفته: این مدل‌ها را می توان در موقعیت‌ها و مجموعه داده‌های مختلف بدون نیاز به تغییرات قابل توجه اعمال کرد.

---

[1] strict distributional assumptions

[2] Random Forest

با این حال، این مدل‌های یادگیری ماشین سنتی در ثبت کامل ویژگی‌های زمانی[3] و وابستگی‌های بلندمدت[4] در داده‌های سری زمانی با چالش‌هایی مواجه هستند که پیشرفت‌های بیشتر در دقت پیش‌بینی را محدود می‌کند.

رویکردهای آماری در پیش‌بینی باد عمدتاً شامل روش‌های پیش‌بینی سری‌های زمانی هستند.

برخی از نمونه های پرکاربرد عبارتند از:

میانگین متحرک یکپارچه خود رگرسیون (ARIMA): این روش مقادیر گذشته را در یک سری زمانی تجزیه و تحلیل می کند تا مقادیر آینده را پیش بینی کند.

ناهمگونی مشروط خود بازگشتی تعمیم یافته (GARCH): این روش برای مدل سازی و پیش بینی نوسانات داده های سری زمانی استفاده می شود، که به ویژه برای داده های انرژی باد که اغلب نوسانات بالایی را نشان می دهند، مرتبط است.

3-1-1. مدل‌های یادگیری عمیق

همانطور که گفته شد، مدل‌های یادگیری ماشین سنتی مانند SVM، Random Forest و XGboost در پیش‌بینی نیروی باد استفاده می‌شوند، اما با این حال در پیش‌بینی موثر ویژگی‌های زمانی و وابستگی‌های طولانی مدت عملکرد قابل قبولی ندارند. مدل های یادگیری عمیق مانند GRU، LSTM و CNN بر این محدودیت ها به روش های زیر غلبه می کنند:

شبکه‌های عصبی حافظه کوتاه‌مدت (LSTM): شبکه‌های LSTM به‌طور خاص طراحی شده‌اند تا مشکل شیب ناپدید شونده[5] را که در RNNهای سنتی حل کنند. RNN های سنتی در مواجهه با دنباله‌های طولانی با این مشکل مواجه می‌شود. این مدل‌ها با ترکیب سلول‌های حافظه[6] و مکانیسم‌های دروازه‌ای[7] که به شبکه اجازه می‌دهد به‌طور انتخابی اطلاعات را برای مدت‌های طولانی به خاطر بسپارد یا فراموش کند، مشکل را برطرف کرده‌اند. بنابراین LSTMها برای ثبت وابستگی‌های بلندمدت در داده‌های انرژی باد مناسب هستند پیش‌بینی‌های دقیق‌تر ارائه می‌دهند.

شبکه‌های عصبی واحد بازگشتی دروازه‌ای (GRU): مشابه LSTM ها، GRU ها از مکانیسم های دروازه ای برای کنترل جریان اطلاعات در داخل شبکه استفاده می کنند و آنها را قادر می سازد تا

---

[3] temporal characteristics

[4] long-term dependencies

[5] vanishing gradient problem

[6] memory cells

[7] gating mechanisms

وابستگی های طولانی مدت را در داده های متوالی دریافت کنند. با این حال، GRU ها معماری ساده تری نسبت به LSTM ها دارند، و آنها را از نظر محاسباتی کارآمدتر می کند و در عین حال عملکرد خوبی را در ثبت الگوهای زمانی در داده های انرژی باد حفظ می کند.

شبکه های عصبی کانولوشنال (CNN): این شبکه‌ها برای پردازش تصویر استفاده می‌شدند و در ادامه برای تجزیه و تحلیل سری های زمانی نیز مورد استفاده واقع شده‌اند. با اعمال فیلترهای کانولوشنال بر روی داده های ورودی، CNN ها می توانند به طور موثر ویژگی ها و الگوهای محلی را در سری های زمانی استخراج کنند. این قابلیت به آن‌ها اجازه می‌دهد تا وابستگی‌ها و نوسانات کوتاه‌مدت در داده‌های انرژی باد را ثبت کنند و دقت پیش‌بینی‌های کوتاه‌مدت را افزایش دهند.

1-1-4. مکانیزم خودتوجهی[8] در شبکه مبدل[9]

مکانیسم خودتوجهی یکی از اجزای مهم شبکه ترانسفورماتور است که آن را قادر می‌سازد از شبکه‌های عصبی بازگشتی (RNN) سنتی مانند LSTM و GRU عملکرد بهتری داشته‌باشند. این عملکرد روی داده‌های سری زمانی با وابستگی‌های طولانی[10] و همبستگی‌های محلی[11] مانند پیش‌بینی نیروی باد ملموس‌تر است.

گرفتن وابستگی‌های دوربرد[12]: برخلاف RNNها که داده‌های متوالی را به‌صورت تکراری پردازش می‌کنند و برای حفظ اطلاعات در توالی‌های طولانی کاربرد دارند، این مکانیزم به مدل اجازه می‌دهد تا همه موقعیت‌های داخل دنباله را به طور همزمان در نظر بگیرد. یا به عبارتی می تواند مستقیماً روابط بین نقاط داده ای را که از نظر زمانی دور از هم قرار دارند را یاد بگیرد. این قابلیت برای پیش بینی دقیق مسائلی مانند نیروی باد که می تواند تحت تأثیر الگوهای آب و هوا و سایر عواملی باشد که در دوره های طولانی آشکار می شوند، حائز اهمیت است.

استخراج موثر همبستگی‌های محلی[13]: با اینکه بررسی وابستگی‌های دوربرد در داده به دقت مدل کمک می‌کند، این مکانیزم به طور مؤثری همبستگی‌های محلی را در داده‌های سری زمانی شناسایی کرده و از آنها استفاده می‌کند. با توجه به روابط بین نقاط داده نزدیک، مدل می تواند روندها و نوسانات کوتاه مدتی را که به دقت پیش بینی کلی کمک می کند، ثبت کند.

---

[8] self-attention mechanism

[9] Transformer Network

[10] long dependencies

[11] local correlations

[12] Capture long-range dependencies

[13] Efficiently extract local correlations

پردازش موازی برای سرعت[14]: قابلیت انجام محاسبات به صورت موازی باعث می شود شبکه ترانسفورماتور به طور قابل توجهی سریعتر از RNN هایی که به پردازش متوالی نیاز دارند، آموزش داده شود.

مکانیسم خودتوجهی، شبکه Transformer را قادر می‌سازد تا با در نظر گرفتن وابستگی‌های دوربرد و همبستگی‌های محلی، یک نمایش جامع از سری‌های زمانی ایجاد کند. در نتیجه مدل به پیش‌بینی‌های دقیق‌تر و قوی‌تری می‌رسد. این نتایج مخصوصا روی پیش‌بینی نیروی باد که در آن فعل و انفعالات پیچیده و چند مقیاسی بر تولید برق آینده تأثیر می‌گذارد، قابل مشاهده می‌باشد.

1-1-5. تابع خطای Huber

استفاده از این تابع خطا در مسائلی که مشکل نقاط پرت در داده دارند، باعث آموزش مدل‌های پایدارتر و مستحکم‌تر می‌شود. در ادامه برخی ویژگی‌های این تابع را بررسی می‌کنیم:

محاسبه خطای تطبیقی[15]: این تابع خطا بر خلاف توابع خطای سنتی مانند MSE که تمامی خطاها را به صورت درجه دوم جریمه می‌کند، از یک رویکرد ترکیبی استفاده می‌کند. این تابع برای خطاهای کوچک‌تر که اندازه‌ای زیر یک آستانه تعریف‌شده δ دارند، همانند تابع خطای MSE رفتار می‌کند و میزان بزرگی خطا را به حداقل می‌رساند. تابع برای خطاهای بزرگتر که اندازه‌ای بیشتر از مقدار δ دارند، مانند تابع خطای خطی[16] عمل می‌کند. این رویکرد باعث می‌شود تاثیر نقاط پرت کاهش داده شود. این رویکرد روی مجموعه داده‌هایی که در ماهیت خود داده پرت زیادی دارند مفید است، چرا که تاکید بیش از حد روی خطای داده‌های پرت می‌تواند آموزش مدل را منحرف کند و دقت پیش‌بینی را کاهش دهد.

کاهش حساسیت به نقاط دورافتاده[17]: داده‌های بادهای فراساحلی به دلیل تغییرات ناگهانی در سرعت و جهت باد، عملکرد نادرست تجهیزات یا خطاهای اندازه‌گیری، اغلب حاوی نقاط پرت هستند. تابع خطای هوبر نسبت به موارد پرت رفتار خطی نشان داده و در برابر خطاهای بزرگ مستحکم است. مدل با استفاده از این تابع روی مجموعه داده با نقاط پرت و غیر قابل پیش‌بینی نسبتا زیاد، پایدارتر و قابل اعتمادتر عمل می‌کند.

---

[14] Parallel processing for speed

[15] Adaptive Loss Calculation

[16] linear loss function

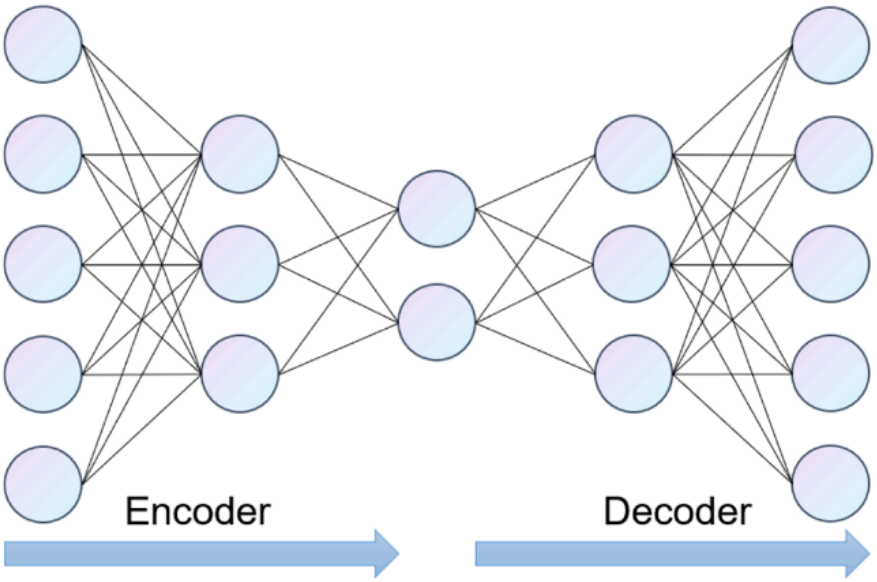
[17] Reduced Sensitivity to Outliers

بهبود پایداری در حین بهینه سازی[18]: تابع ضرر هوبر در کل دامنه خود پیوسته و قابل تمایز است. در حالی که تابع ضرر مطلق در نقطه صفر مشتق‌ناپذیر است. این ویژگی تابع به smoth بودن بهینه‌سازی gradient descent کمک می‌کند و به مدل اجازه می‌دهد تا پارامترهای بهینه را بدون ایجاد اختلال در تغییرات ناگهانی در مشتق تابع خطا، به طور موثر پیدا کند. این ویژگی امکان بهینه سازی نرم تر، پایداری و همگرایی کلی مدل را افزایش می دهد.

1-2. آماده‌سازی

1-2-1. ساختار Autoencoder

ساختار Autoencoder یک الگوی یادگیری بدون نظارت[19] بوده و برای بازسازی و حذف نویز داده‌های نیروی باد دریایی به منظور بهبود دقت پیش‌بینی‌ها استفاده شده‌است.



شکل 1. نمودار شماتیک از یک Autoencoder

معماری این عصبی پیچیده به دو جزء اساسی تقسیم می شود: رمزگذار و رمزگشا.

بخش رمزگذاری یا encoder ای معماری رابطه ریاضی $f: R^d \rightarrow R^h$ را پیاده می‌کند که داده ورودی در دامنه $x \in R^d$ می‌باشد. در جهت مقابل، در بخش رمزگشا یا decoder این معماری، که

[18] Improved Stability During Optimization

[19] unsupervised learning paradigm

رابطه ریاضی آن به صورت $g: R^h \rightarrow R^d$ قابل نمایش است، بردار نمایش ثانویه از داده را به فضای برداری اولیه آن برمی‌گرداند. در نهایت فرآیند به سنتز یک ورودی بازسازی شده ختم می‌شود که معادله‌ی آن به صورت $\hat{x} = g(f(x))$ است.

در فرایند یادگیری یک Autoencoder، بهینه‌سازی پارامترها بر اساس کمینه کردن مقدار خطای تابع $L(x, \hat{x})$ می‌باشد که در آن فاصله بین داده ورودی و داده‌ی خروجی اندازه گیری می‌شود. معمولا در این مسئله از تابع خطای mean square error loss استفاده می‌شود که فرمول آن به شرح زیر است:

$$L(x, \hat{x}) = (x - \hat{x})^2$$

1-2-2. مکانیزم توجه

مکانیسم توجه شبکه ترانسفورمر را قادر می سازد تا هنگام پردازش آن به طور انتخابی بر روی بخش‌های خاصی از دنباله ورودی تمرکز کند. این به مدل اجازه می دهد تا مرتبط ترین اطلاعات را برای کار پیش بینی یاد بگیرد و اولویت بندی کند.

Multi-head Attention: مکانیزم توجه چند سر باعث می‌شود توالی ورودی در چندین "سر" پیش بینی شود. هر کدام از این چند بخش بر جنبه‌های مختلف داده‌ها تمرکز می‌کنند. در نهایت با ترکیب آن‌ها به درک ظریف‌تر و غنی‌تری از دنباله ارائه می‌شود.

Capturing Dependencies: مکانیزم توجه به شناسایی روابط بین نقاط داده در دنباله، صرف نظر از فاصله‌ی آنها معروف است. این ویژگی تاثیر مهمی روی گرفتن وابستگی‌های دوربرد و همبستگی‌های محلی دارد. وابستگی‌های دور برد جاهایی رخ می‌دهد که رویدادهای گذشته بر نیروی باد آینده تاثیرگذار باشد. همبستگی‌های محلی در داده زمانی وجود دارد که نقاط داده‌ی نزدیک الگوهای مشابهی را نشان بدهند.

1-2-3. رمزگذاری موقعیت[20]

با اینکه مکانیسم توجه همه موقعیت های درون دنباله را به طور همزمان در نظر می‌گیرد، اما به طور ذاتی ترتیب نقاط داده را درک نمی کند. رمزگذاری موقعیتی این دست از اطلاعات را بررسی کرده و مدل را قادر می‌سازد بین نقاط داده بر اساس موقعیت آن‌ها در دنباله نقاط تمایز قائل شود.

[20] Positional Encoding

Encoding Temporal Order: رمزگذاری موقعیتی در کنار encoding ورودی قرار گرفته و در نتیجه اطلاعات مربوط به موقعیت نسبی نقاط ارائه می‌شود. در نتیجه مدل می‌آموزد که ترتیب رویدادها چگونه است و چه تاثیری روی پیش‌بینی‌ها می‌گذارد. در نتیجه مدل به ترتیب داده‌ها در فرآیند یادگیری اهمیت می‌دهد.

Improved Accuracy: با ترکیب اطلاعات موقعیتی، شبکه ترنسفورمر می‌تواند دینامیک زمانی داده‌های انرژی باد را بهتر درک کند. این مورد در پیش‌بینی‌های چند مرحله‌ای دقت خوبی می‌دهد.

1-2-4. رفتار تابع خطای Huber

در بخش 1-1-5 به توضیح این بخش پرداخته‌ایم.

1-2-5. الگوریتم Slime mould

الگوریتم بهینه‌سازی "میکروب‌های لجن" (Slime Mould Optimization Algorithm - SMOA) یک تکنیک بهینه‌سازی اکتشافی[21] است که از رفتار جستجوی کپک‌های لجن در طبیعت الهام گرفته شده است. در این الگوریتم، هر "میکروب" یا عامل نمایانگر یک مجموعه از پارامترهای مدل است که به طور تصادفی از فضای جستجوی تعریف‌شده انتخاب می‌شود. سپس، این عوامل حرکت می‌کنند و تلاش می‌کنند به مجموعه‌ای از پارامترها برسند که بیشترین کارایی را داشته باشد. الگوریتم حرکت عامل‌ها را از روی معیارهای عملکرد (fitness) آن‌ها تعیین می‌کند.

نحوه عملکرد الگوریتم:

1. **ایجاد عامل‌ها (Agents)**: ابتدا تعدادی عامل (یا میکروب) به طور تصادفی از فضای جستجو انتخاب می‌شوند.
2. **ارزیابی عملکرد (Fitness Evaluation)**: برای هر عامل، مدل با پارامترهای انتخابی آن آموزش داده می‌شود و عملکرد آن با استفاده از داده‌های اعتبارسنجی ارزیابی می‌شود.
3. **حرکت عوامل**: سپس عوامل بر اساس میزان موفقیت‌شان (fitness) حرکت می‌کنند و تلاش می‌کنند به پارامترهایی برسند که بهترین عملکرد را دارند.
4. **به‌روز رسانی بهترین عامل**: عامل‌هایی که بهترین عملکرد را دارند به سمت همدیگر جذب می‌شوند و این فرآیند ادامه می‌یابد.

[21] Heuristic optimization technique

پارامترهای ورودی:

1. **hyperparameter_space (فضای پارامترهای بهینه‌سازی)**: این دیکشنری شامل مقادیر ممکن برای هر پارامتر است که الگوریتم می‌تواند آن‌ها را جستجو کند. این پارامترها شامل موارد زیر هستند:
   - "dropout_rate": نرخ افت (Dropout) که به طور معمول بین 0.1 تا 0.5 تغییر می‌کند.
   - "num_heads": تعداد سرهای توجه (Attention heads) که مقادیر ممکن آن 2، 4، 8 و 16 است.
   - "key_dim": ابعاد کلیدها (Key dimension) که مقادیر آن شامل 32، 64 و 128 است.
2. **fitness_function (تابع ارزیابی عملکرد)**: این تابع به منظور ارزیابی عملکرد مدل با پارامترهای مختلف طراحی شده است. در اینجا، مدل با پارامترهای انتخاب‌شده آموزش داده می‌شود و سپس با استفاده از داده‌های اعتبارسنجی، میزان خطا (در اینجا MSE یا خطای مربعات میانگین) محاسبه می‌شود.
3. **num_agents (تعداد عامل‌ها)**: تعداد عامل‌هایی که در فضای جستجو حرکت می‌کنند. این مقدار می‌تواند به تعداد دلخواه تنظیم شود، اما تعداد زیادتر عامل‌ها زمان محاسبات را افزایش می‌دهد که در این مسئله از تعداد 5 استفاده کرده‌ایم.
4. **max_iter (تعداد تکرارها)**: تعداد تکرارهایی که الگوریتم برای بهینه‌سازی اجرا می‌کند. این پارامتر مشخص می‌کند که الگوریتم چند بار جستجو و به‌روزرسانی عامل‌ها را انجام دهد که در این مسئله به 20 ست کرده ایم.
5. **input_dim (ابعاد ورودی داده‌ها)**: ابعاد داده‌های ورودی به مدل. این پارامتر مشخص می‌کند که داده‌ها دارای چه تعداد ویژگی و طول دنباله هستند.

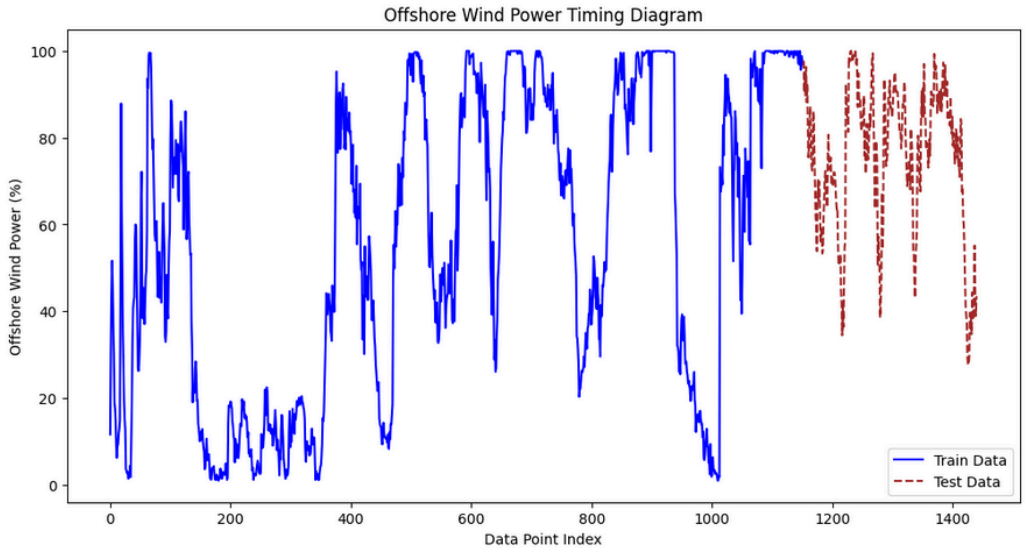
روند اجرای الگوریتم:

1. ابتدا تعداد مشخصی عامل به طور تصادفی از فضای پارامترهای بهینه‌سازی انتخاب می‌شوند.
2. برای هر عامل، عملکرد آن با استفاده از داده‌های آموزشی و اعتبارسنجی ارزیابی می‌شود.
3. عوامل در فضای جستجو حرکت می‌کنند و تلاش می‌کنند به بهترین مجموعه پارامترها دست یابند.

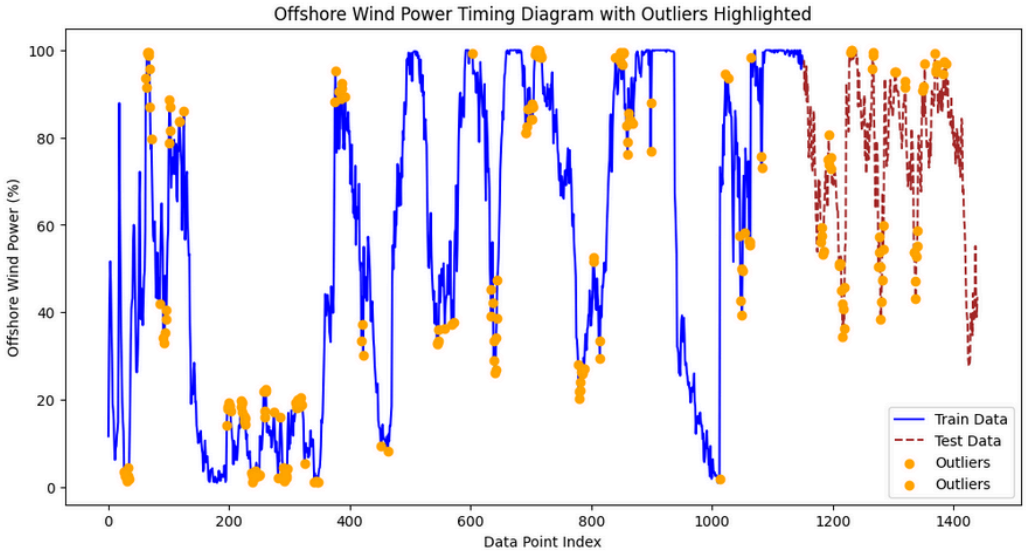
4. فرآیند ارزیابی و به‌روز رسانی عامل‌ها ادامه می‌یابد تا الگوریتم به بهترین نتیجه برسد.

3-1. روش‌شناسی و نتایج

بخشی از داده‌ها که در مقاله توضیح داده شده است را انتخاب کنید و به شکل زیر می‌رسیم.



به کمک روشی که در مقاله ذکر شده، داده‌های پرت را شناسایی کرده و به شکل زیر می‌رسیم.



این روش به صورت کد زیر پیاده شده است:

```python
def calculate_rolling_statistics(dataframe, window_size):
    dataframe_column = dataframe['y (% relative to rated power)']
    rolling_mean = dataframe_column.rolling(window=window_size, center=True).mean()
    rolling_variance = dataframe_column.rolling(window=window_size, center=True).var()
    return rolling_mean, rolling_variance

def identify_outliers(dataframe, rolling_mean, rolling_variance, threshold=1):
    dataframe_column = dataframe['y (% relative to rated power)']
    std_dev = np.sqrt(rolling_variance)
    upper_limit = rolling_mean + (threshold * std_dev)
    lower_limit = rolling_mean - (threshold * std_dev)

    outliers = (dataframe_column > upper_limit) | (dataframe_column < lower_limit)
    return outliers
```

- این کد ابتدا با استفاده از تابع calculate_rolling_statistics آمار متحرک (میانگین و واریانس) یک سری داده را محاسبه می‌کند.
- سپس با تابع identify_outliers نقاط پرت را بر اساس فاصله آن‌ها از میانگین و انحراف معیار مشخص‌شده پیدا می‌کند. این نقاط پرت می‌توانند برای تحلیل داده یا حذف نویزها در پیش‌پردازش داده مفید باشند.

داده‌ها را با نسبت 80 به 20 به دو دسته آموزش و ارزیابی تقسیم کرده‌ایم.

پیش پردازش انجام شده بر روی داده‌ها به صورت نرمال‌سازی پیاده می‌شود. این فرآیند به بهبود کارایی الگوریتم‌های یادگیری ماشین کمک می‌کند و بیشتر الگوریتم‌ها با داده‌های نرمال‌سازی‌شده بهتر عمل می‌کنند.

MinMaxScaler ، داده‌های آموزشی را برای نرمال‌سازی آموزش می‌دهد و آن‌ها را مقیاس‌بندی می‌کند. در ادامه داده‌های تست را با استفاده از مقادیر حداقل و حداکثر محاسبه‌شده از داده‌های آموزشی مقیاس‌بندی می‌کند.

همانطور که صورت مسئله ما تعریف شده است، مقدار نیروی باد متغیری است که به دنبال پیشبینی آن هستیم. یعنی هدف ما این است مدلی آموزش دهیم که مقدار این متغیر را بر اساس بقیه متغیر ها تخمین بزند. متغیر y متغیر وابسته بوده و بقیه متغیرها مستقل می‌باشند.

در این بخش به پیاده‌سازی پنجره متحرک می‌پردازیم. در این کد، یک پنجره متحرک برای ایجاد داده‌های ورودی و خروجی برای مدل پیش‌بینی استفاده شده است. تابع create_sliding_window به منظور ایجاد داده‌های ورودی و خروجی با استفاده از یک پنجره

متحرک به کار می‌رود. این تابع متغیر prediction_size را به عنوان تعداد پیش‌بینی‌ها برای هر پنجره در نظر گرفته است که مقدار پیش‌فرض آن 1 است. متغیر ورودی window_size نیز اندازه پنجره برای استخراج داده‌های ورودی را مشخص می‌کند. مقدار پیش فرض این متغیر هم 144[22] است، مشابه مقاله. در نتیجه این عمل، به مدل کمک می‌شود داده‌های سری زمانی به وابستگی زمانی بین داده‌ها را بهتر درک کند.

با اعمال یک حلقه برای هر نقطه از داده‌ها، داده‌های ورودی (X) و خروجی (y) را تولید می‌کند که X شامل داده‌های ورودی است که اندازه آن برابر با window_size است. y شامل داده‌های پیش‌بینی است که به اندازه prediction_size انتخاب می‌شود.

در بخش **اصلاح شکل داده‌ها**، برای داده‌های ورودی X_train و X_test، بعد آخر آن‌ها که آخرین ویژگی‌ها را شامل می‌شود، حذف می‌شود (corected_X_train و corected_X_test). برای داده‌های خروجی، تنها آخرین ویژگی برای پیش‌بینی انتخاب می‌شود (corected_y_train و corected_y_test).

در بخش **تسطیح داده‌ها**، داده‌های ورودی به شکلی مسطح (flatted) تبدیل می‌شوند تا بتوانند به مدل MLP، داده شوند. این فرآیند X_train_flat و X_test_flat داده‌های ورودی را به صورت یک‌بعدی مسطح می‌کنند. علت این مورد هم این است که این مدل دیتای Sequential را درک نمیکند. در نهایت روابط Fully Connected داده‌ی متوالی تسطیح شده را وزن‌هایش را به نحوی تنظیم می‌کند که ارتباط بین قسمت‌ها مختلف به خوبی تشخیص داده شود.

در بخش **افزایش ابعاد داده‌ها برای ورودی به مدل‌های توالی**، برای مدل‌هایی که نیاز به ورودی‌های توالی‌دار (sequence) دارند، مانند شبکه‌های عصبی بازگشتی (RNNs)، ابعاد داده‌ها گسترش می‌یابد و داده‌های ورودی به فرم توالی با استفاده از np.expand_dims تبدیل می‌شوند (X_train_seq و X_test_seq).

در ترتیب پیش‌پردازش‌ها هم باید تقدم تقسیم بندی مجموعه داده به آموزش و ارزیابی به نرمال‌سازی رعایت شود، چرا که نرمال‌سازی باید بر اساس مقادیر مجموعه داده آموزشی انجام شود. اگر ابتدا کل داده‌ها نرمال‌سازی شوند و سپس تقسیم‌بندی انجام گیرد، اطلاعات مجموعه ارزیابی به

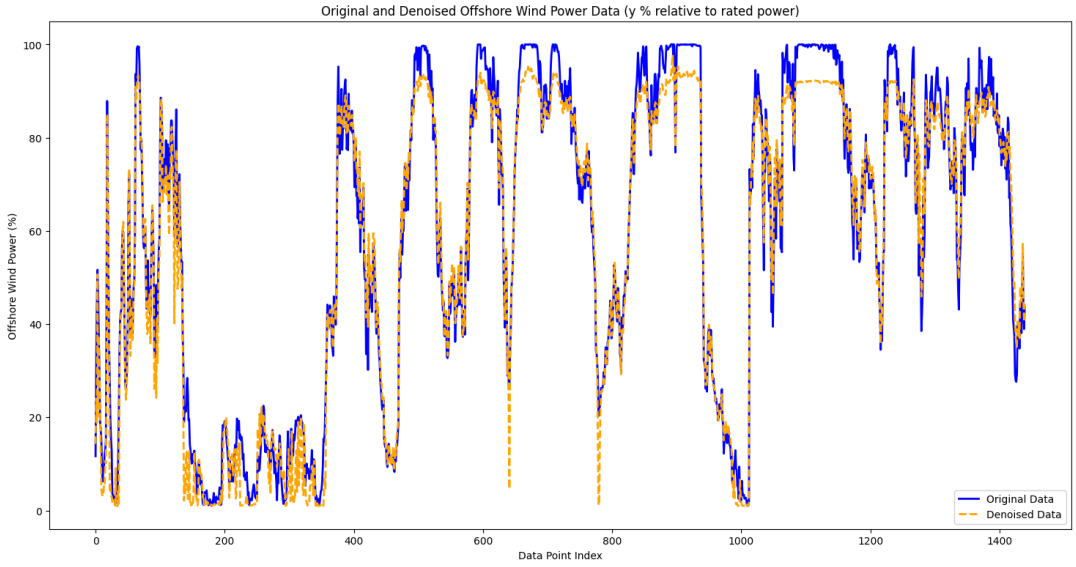
[22] این مقدار معادل 24 ساعت از زمان است.

طور غیرمستقیم در فرآیند نرمال‌سازی وارد مدل می‌شود. این کار باعث می‌شود که مدل به داده‌های ارزیابی دسترسی داشته باشد و ارزیابی مدل به درستی انجام نشود.

به عبارت دیگر، **نرمال‌سازی تنها بر اساس مقادیر مجموعه آموزشی** صورت می‌گیرد تا از نشت اطلاعات (Data Leakage) جلوگیری شود و مدل عملکرد واقعی خود را روی داده‌هایی که قبلاً ندیده است، نشان دهد.

پس از انجام نرمال‌سازی و آماده‌سازی داده‌ها، فرآیند آموزش مدل آغاز می‌شود و سپس عملکرد آن بر روی مجموعه ارزیابی بررسی خواهد شد.

حال برای کم کردن اثر این داده‌های پرت، داده‌هایمان را denoise می‌کنیم. برای این منظور یه AutoEncoder طراحی کرده‌ایم. این فرآیند در 50 ایپاک و با بچ سایز 64 صورت می‌گیرد. در نهایت مقایسه‌ای روی داده‌های اولیه و داده‌های denoise شده انجام می‌دهیم.



حال به بررسی فرآیند آموزش مدل‌ها می‌پردازیم.

1-3-1. در حالت Single Step

نتیجه این مدل‌ها به شرح زیر است:

در این مدل‌ها در حالت **Single Step Prediction**، نتایج مدل‌ها نشان‌دهنده عملکرد نسبی آنها در پیش‌بینی مقدار t+1 است.

Model	Loss Function	MAE	MAPE	RMSE	R2
MLP	MSE	0.057816	4.284032	0.083469	0.932396
MLP	Huber	0.052013	2.634830	0.076833	0.942717
RNN	MSE	0.038581	5.835179	0.059015	0.966205
RNN	Huber	0.038287	4.494176	0.060180	0.964857
Transformer	MSE	0.134872	19.793905	0.170783	0.716980
Transformer	Huber	0.131675	13.636563	0.164747	0.736631

Model	Loss Function	MAE	MAPE	RMSE	R2
MLP	MSE	0.057816	4.284032	0.083469	0.932396
MLP	Huber	0.052013	2.634830	0.076833	0.942717
RNN	MSE	0.038581	5.835179	0.059015	0.966205
RNN	Huber	0.038287	4.494176	0.060180	0.964857
Transformer	MSE	0.134872	19.793905	0.170783	0.716980
Transformer	Huber	0.131675	13.636563	0.164747	0.736631

1-3-2. اعمال Slime Mould برای بهینه‌سازی

در این بخش، از الگوریتم **Slime Mould** برای بهینه‌سازی پارامترهای مدل **Transformer** در حالت **Single Step Prediction** استفاده شده است. هدف اصلی بهینه‌سازی، یافتن مقادیر بهینه برای پارامترهای کلیدی مدل جهت بهبود دقت پیش‌بینی و کاهش مقدار خطا بوده است.

با اعمال این بخش بر روی مدل Transformer با حالت Single Step، به پارامترهای زیر رسیده‌ایم:

dropout_rate	num_heads	key_dim	Best fitness
0.4	**16**	**64**	**0.04604**

توضیح نحوه عملکرد و ساختاری کد استفاده شده در بخش توضیح آن آورده شده است. پارامترهای به دست آمده، مقدار فیتنس بهتری ارائه داده است.

1-3-3. در حالت Multi Step با t+4

در این مدل، پیش‌بینی 40 دقیقه پس از داده‌های فعلی مدنظر است.

Model	Loss Function	MAE	MAPE	RMSE	R2
MLP	MSE	0.062541	19.031906	0.092138	0.871083
MLP	Huber	0.101779	24.949947	0.130313	0.772908
RNN	MSE	0.058035	18.459522	0.088064	0.872939
RNN	Huber	0.061214	19.851849	0.090318	0.870929
Transformer	MSE	0.124282	32.878239	0.159917	0.742826
Transformer	Huber	0.130121	35.564049	0.168449	0.712509

Model	Loss Function	MAE	MAPE	RMSE	R2
MLP	MSE	0.062541	19.031906	0.092138	0.871083
MLP	Huber	0.101779	24.949947	0.130313	0.772908
RNN	MSE	0.058035	18.459522	0.088064	0.872939
RNN	Huber	0.061214	19.851849	0.090318	0.870929
Transformer	MSE	0.124282	32.878239	0.159917	0.742826
Transformer	Huber	0.130121	35.564049	0.168449	0.712509

4-3-1. در حالت Multi Step با t+8

در این مدل، پیش‌بینی 80 دقیقه پس از داده‌های فعلی مدنظر است.

Model	Loss Function	MAE	MAPE	RMSE	R2
MLP	MSE	0.063038	19.546246	0.092672	0.880559
MLP	Huber	0.074419	27.481785	0.102205	0.863739
RNN	MSE	0.059675	23.997797	0.090036	0.882171
RNN	Huber	0.070327	25.491070	0.100069	0.866813
Transformer	MSE	0.130215	33.939487	0.170030	0.711863
Transformer	Huber	0.141552	61.913929	0.183364	0.658616

Model	Loss Function	MAE	MAPE	RMSE	R2
MLP	MSE	0.063038	19.546246	0.092672	0.880559
MLP	Huber	0.074419	27.481785	0.102205	0.863739
RNN	MSE	0.059675	23.997797	0.090036	0.882171
RNN	Huber	0.070327	25.491070	0.100069	0.866813
Transformer	MSE	0.130215	33.939487	0.170030	0.711863
Transformer	Huber	0.141552	61.913929	0.183364	0.658616

4-3-1. در حالت Multi Step با t+16

در این مدل، پیش‌بینی 160 دقیقه پس از داده‌های فعلی مدنظر است.

Model	Loss Function	MAE	MAPE	RMSE	R2
MLP	MSE	0.077074	26.396961	0.106483	0.858328
MLP	Huber	0.094278	25.220221	0.125892	0.812520
RNN	MSE	0.058133	23.854557	0.085988	0.889228
RNN	Huber	0.062896	24.527620	0.088666	0.884119
Transformer	MSE	0.148563	45.680786	0.189027	0.650806
Transformer	Huber	0.146758	35.698204	0.186110	0.660262

Model	Loss Function	MAE	MAPE	RMSE	R2
MLP	MSE	0.077074	26.396961	0.106483	0.858328
MLP	Huber	0.094278	25.220221	0.125892	0.812520
RNN	MSE	0.058133	23.854557	0.085988	0.889228
RNN	Huber	0.062896	24.527620	0.088666	0.884119
Transformer	MSE	0.148563	45.680786	0.189027	0.650806
Transformer	Huber	0.146758	35.698204	0.186110	0.660262

1-3-5. مقایسه مدل‌های Multi Step

همانطور که مشخص است، هر چقدر مقدار زمان مورد انتظار برای پیش‌بینی کمتر باشد، مدل‌ها نتایج به نسبت بهتری ارائه داده‌اند. علت این مدل هم این است که در پیش‌بینی‌های چندمرحله‌ای (Multi-Step) با افق زمانی بلندتر دقت کمتری دارند، به چندین عامل اساسی برمی‌گردد:

1. **انباشت خطا (Error Accumulation):**
   در پیش‌بینی چندمرحله‌ای، پیش‌بینی‌های قبلی به عنوان ورودی برای پیش‌بینی مراحل بعدی استفاده می‌شوند. این موضوع باعث می‌شود که خطاهای کوچک در مراحل اولیه، با گذشت زمان تجمع یابند و دقت مدل را کاهش دهند.
2. **وابستگی‌های بلندمدت پیچیده:**
   با افزایش فاصله زمانی (مانند t+16)، مدل‌ها باید وابستگی‌های زمانی طولانی‌تری را یاد بگیرند. این وظیفه برای مدل‌هایی مثل MLP و RNN که توانایی کمتری در مدل‌سازی وابستگی‌های بلندمدت دارند، دشوارتر است.
3. **کاهش سیگنال قابل پیش‌بینی:**
   با افزایش افق زمانی، تاثیر الگوهای قابل پیش‌بینی موجود در داده‌ها کاهش می‌یابد و

مدل‌ها باید با نویز بیشتری در داده‌ها مقابله کنند. این مسئله به ویژه برای مدل‌های ساده‌تر مانند MLP چالش‌برانگیزتر است.

4. **کمبود اطلاعات هدف:**
   مدل‌ها به طور طبیعی برای افق‌های زمانی کوتاه‌تر (مانند t+4) اطلاعات کافی در داده‌های تاریخی دارند. اما در بازه‌های بلندتر، اطلاعات هدف کمتر شده و مدل‌ها باید به شکلی غیرمستقیم روابط را استنتاج کنند.
5. **محدودیت‌های معماری مدل‌ها:**
   - **MLP** به دلیل نداشتن حافظه داخلی و ارتباط مستقیم بین ورودی و خروجی، نمی‌تواند وابستگی‌های زمانی را به خوبی مدل‌سازی کند.
   - **RNN** با وجود حافظه داخلی، در یادگیری وابستگی‌های بلندمدت دچار مشکل می‌شود (مشکل ناپایداری گرادیان).
   - **Transformer** به دلیل استفاده از مکانیزم Self-Attention، توانایی بیشتری در مدل‌سازی این وابستگی‌ها دارد، اما همچنان با افزایش افق پیش‌بینی، دقت کاهش می‌یابد.

6-3-1. مقایسه مدل‌های مختلف با معماری‌ها منحصر به فرد

همانطور که در بخش‌های اولیه ذکر شد، این مدل‌ها معماری‌ها، نحوه‌ی عملکرد متفاوتی دارند. همانطور که در تمامی بخش‌ها قابل مشاهده است، در تمامی موارد مدل RNN نسبت به مدل MLP عملکرد بهتری داشته و MLP نیز نسبت به مدل transformers عملکرد بسیار بهتری داشته است و مقادیر خطای کمتری ثبت کرده است. علت عملکرد بسیار ضعیف transformers را میتوان ابتدا به ویژگی دادگان مورد استفاده ارتباط داد که وابستگی های زمانی موجود در این تسک خاص احتمالا وابستگی زمانی بلند مدتی نبوده و از آنجا که transformers بهترین عملکرد خود را در تشخیص وابستگی های زمانی بلند مدت تر نشان میدهد تا مواردی مانند این تسک که وابستگی های زمانی طولانی کمتری وجود دارد امکان بهره‌وری از توانایی های مدل را کاهش میدهد همچنین امکان دارد که مکانیزم اتنشن در جایگاه درستی متمرکز نشده باشد که با اپتیمایز کردن این مورد امکان بهبود عملکرد مدل وجود دارد. در مورد عملکرد بهتر RNN از MLP نیز به دلیل توانایی مدل RNN در تشخیص الگو های زمانی کوتاه مدت تر این مدل هم از وابستگی های میان فیچر ها و فیلد هدف و هم از پترن های زمانی بهره برده و این موضوع باعث میشود که عملکرد نهایی این مدل از MLP که تمامی توالی داده های موجود در یک دیتا پوینت را پشت هم و به صورت فلت دریافت

کرده است و پترن های میان آنها را به کمک وزن های شبکه و نه به کمک حافظه ی موجود از گذشته حفظ کند که این موضوع توانایی کمتر MLP در مدلسازی این پترن های زمانی را توجیه میکند.

دلایل برتری RNN بر MLP در پیش‌بینی سری‌های زمانی:

1. **مدل‌سازی وابستگی‌های زمانی:**
   - معماری **RNN** به صورت ذاتی برای داده‌های ترتیبی و سری‌زمانی طراحی شده است. این مدل با استفاده از حافظه داخلی، می‌تواند وابستگی‌های بین نقاط زمانی مختلف را یاد بگیرد و اطلاعات مربوط به وضعیت‌های قبلی را به مراحل بعدی منتقل کند.
   - معماری **MLP** فقط داده‌ها را به صورت ایستا و مستقل پردازش می‌کند و هیچ اطلاعاتی درباره ترتیب یا وابستگی زمانی بین داده‌ها را در نظر نمی‌گیرد. به همین دلیل، برای پیش‌بینی سری‌های زمانی که ترتیب و تاریخچه داده‌ها بسیار مهم است، مناسب نیست.
2. **حافظه داخلی (Memory):**
   - مدل **RNN** با به‌روزرسانی مداوم حافظه خود، می‌تواند اطلاعات مربوط به گام‌های قبلی را حفظ کرده و برای پیش‌بینی گام فعلی از آن‌ها استفاده کند.
   - **مدل MLP** فاقد هرگونه حافظه داخلی است و فقط از ورودی لحظه‌ای برای پیش‌بینی استفاده می‌کند، بنابراین قادر به درک الگوهای زمانی نیست.
3. **وابستگی‌های کوتاه‌مدت و بلندمدت:**

   مدل **RNN** قادر است هم وابستگی‌های کوتاه‌مدت و تا حدی بلندمدت را مدل‌سازی کند. هرچند ممکن است در وابستگی‌های بسیار طولانی با مشکل ناپایداری گرادیان مواجه شود، اما همچنان بهتر از MLP عمل می‌کند.

   مدل **MLP** به دلیل عدم در نظر گرفتن ترتیب داده‌ها، نمی‌تواند هیچ‌گونه وابستگی زمانی را مدل‌سازی کند.

4. **سازگاری با داده‌های ترتیبی:**
   - **RNN** طراحی شده برای داده‌های ترتیبی مانند سری‌های زمانی، متن، یا سیگنال‌های گفتاری. معماری آن به طور مستقیم برای مسائل سری‌زمانی مناسب است.

- **MLP** بیشتر برای داده‌های ایستا و بدون وابستگی زمانی طراحی شده است (مانند داده‌های جدولی).

5. **کاهش نیاز به ویژگی‌های مهندسی‌شده:**

RNN به دلیل معماری بازگشتی و توانایی پردازش وابستگی‌ها، می‌تواند به صورت خودکار الگوهای زمانی را یاد بگیرد.

MLP معمولاً برای دستیابی به عملکرد بهتر نیازمند ویژگی‌های مهندسی‌شده و از پیش‌پردازش دقیق است.

6. **کاربرد عملی:**

RNN برای مسائلی مانند پیش‌بینی سری‌های زمانی، ترجمه ماشینی، و پردازش زبان طبیعی طراحی شده است.

MLP بیشتر در مسائلی که به وابستگی‌های مکانی و ساختار سلسله‌مراتبی نیازی نیست، استفاده می‌شود.

همانطور که در این نتایج قابل مشاهده است، از آنجایی که داده ورودی تمامی حالات **Denoise** شده و مقدار خوبی از **تاثیر داده‌های پرت** روی مدل کاسته شده است، این دو تابع خطای Huber و MSE نتایج نزدیک به هم ارائه داده‌اند.

پرسش 2 - استفاده از ViT برای طبقه‌بندی تصاویر گلبول‌های سفید

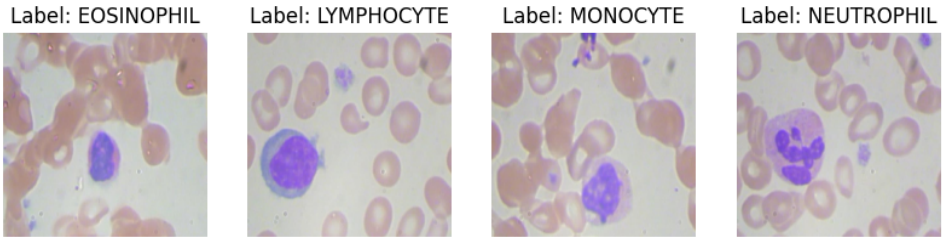
2-1. مقدمه

در این مسئله به کمک مجموعه داده‌ای از انواع گلبول‌های سفید که در اختیار ما قرار داده شده است، به حل مسئله طبقه‌بندی این دسته‌ها می‌پردازیم.

2-2. آماده‌سازی داده‌ها

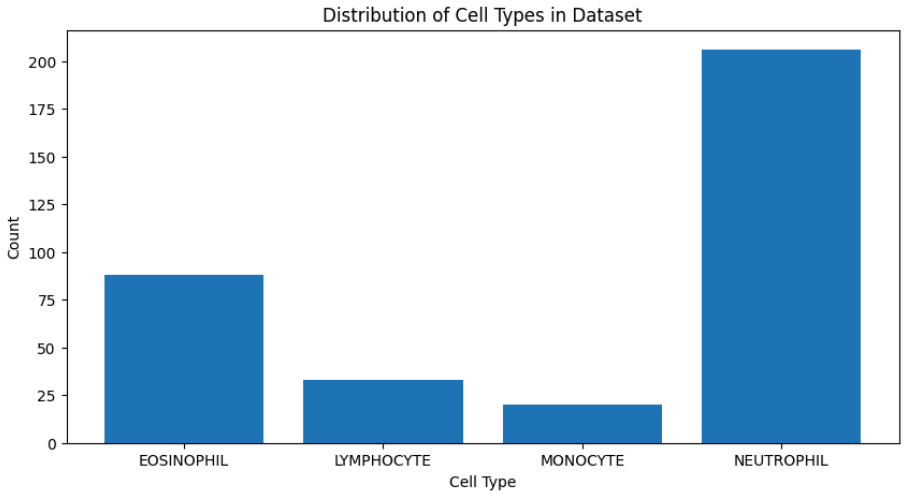
2-2-1. مشاهده تصاویر مجموعه داده

در این بخش ابتدا به مشاهده یک نمونه تصویر از هر دسته گلبول سفید خون در مجموعه داده می‌پردازیم:



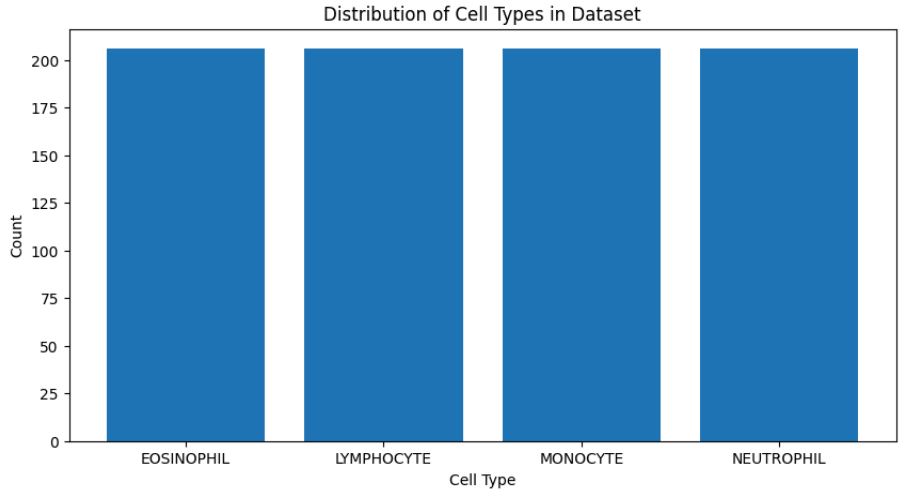
2-2-2. بررسی توزیع کلاس‌های مجموعه داده

حال به بررسی تعداد داده‌های موجود در هر کلاس مجموعه داده با کمک نمودار میله‌ای می‌پردازیم:



شکل 1. نمودار میله‌ای توزیع داده اصلی

همانطور که مشاهده می‌شود، تعداد داده‌ها در قالب تصویر در هر 4 دسته از کلاس‌ها یکسان نبوده داده‌ها متوازن نیستند. این عدم توازن را با به کمک upsampling حل می‌کنیم:



شکل 2. نمودار میله‌ای توزیع داده upsample شده

علت استفاده از updampling، این است که تعداد داده کافی در اختیار مدل‌ها قرار دهیم و همینطور، نیاز است تعداد داده‌ها برابری کنند تا مدل‌ها با تعداد داده‌ی مساوی به خوبی آموزش ببینند.

2-2-3. داده‌افزایی

روش data augmentation استفاده شده به شرح زیر است:

1. ابتدا بالاترین تعداد داده بین 4 کلاس را پیدا می‌کنیم.
2. مجموعه‌ای از تبدیل‌ها (برگردان افقی، چرخش، برش تغییر اندازه، و غیره) را برای اعمال بر روی تصاویر تنظیم می‌کنیم.

ToPILImage	یک آرایه NumPy یا یک تنسور PyTorch را به یک تصویر PIL (کتابخانه تصویربرداری پایتون) تبدیل می کند.
RandomHorizontalFlip	به طور تصادفی تصویر را به صورت افقی با احتمال 0.5 بر می گرداند.
RandomRotation(10)	تصویر را به طور تصادفی در محدوده 10- تا 10+ درجه می چرخاند.
RandomResizedCrop	به طور تصادفی تصویر را به اندازه مشخصی برش می دهد و اندازه آن را به مقدار $(self.img\_size\_w,\ self.img\_size\_h)$ تغییر می دهد.
ToTensor	تصویر PIL را دوباره به یک تنسور PyTorch تبدیل می کند.

علت تغییر سایز تصاویر در بخش RandomResizedCrop این است که تصویر ورودی به مدل ViT، به ابعاد مدنظر مدل وارد شود.

به علت منابع محدود، تعداد تصاویر را به 400 نرسانده‌ایم.

3. این مجموعه را روی داده‌های کلاس‌ها با تعداد کمتر اعمال می‌کنیم تا در نهایت تمامی کلاس‌ها به تعداد داده 206تایی برسند.

2-2-4. تقسیم داده

مطابق خواسته صورت مسئله، مجموعه داده را به دو بخش آموزش و ارزیابی با نسبت‌های 90 به 10 تقسیم کرده‌ایم.

2-3. آموزش مدل‌ها

معماری این مدل‌ها در کد موجود است که شامل تعداد بسیار زیادی از اجزا می‌باشد که از آوردن دقیق آنها در این بخش صرف نظر کرده‌ایم و به جای آن، به توضیح اجزای مهم در بخش‌های زیر می‌پردازیم.[23]

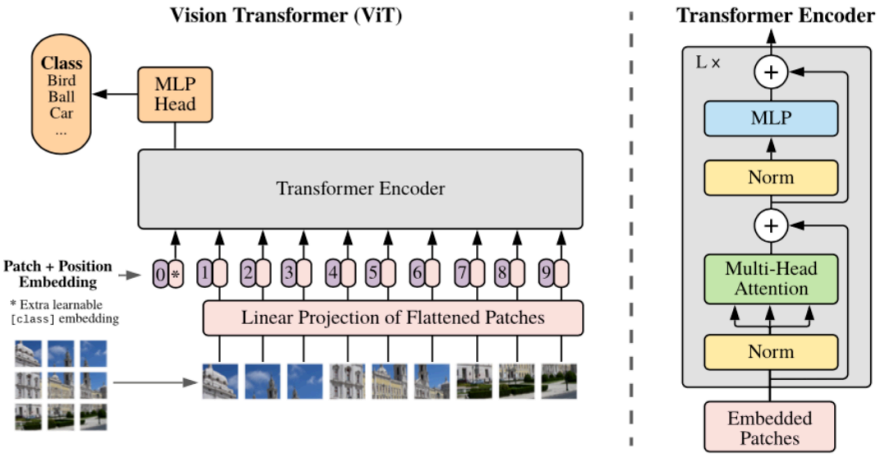
[23] این معماری ها در فایل Q2.ipynb قابل مشاهده می‌باشد.

2-3-1. توضیح مدل ViT

در این مسئله از مدل ViT گوگل استفاده کرده‌ایم. همانطور که در کد مشخص است، تعداد کل پارامترهای این مدل بزرگ از گوگل، حدود 85 هزار پارامتر می‌باشد که عدد بسیار بزرگی‌ست. اما ما در با تعریف حالت‌های زیر، تنها بخشی از آنها را آموزش می‌دهیم.

ویژن ترنسفورمر (ViT) اولین بار در سال 2020 توسط محققان گوگل معرفی شد. این مدل برای اولین بار پردازش تصاویر را با استفاده از ترانسفورمرهای مبتنی بر معماری مشابه به مدل‌های NLP انجام می‌دهد. به‌طور سنتی، پردازش تصاویر با استفاده از شبکه‌های عصبی کانولوشنی (CNN) انجام می‌شد، اما ViT از یک رویکرد متفاوت بهره می‌برد که شامل تقسیم تصویر به قطعات کوچک‌تر (پچ‌ها) و اعمال ترنسفورمر بر روی آن‌ها است.

این مدل ابتدا تصویر ورودی را به بخش‌های کوچکی به نام Patch تقسیم می‌کند. هر Patch پس از فرآیند مسطح‌سازی به یک بردار (Vector) تبدیل می‌شود و به همراه Position Embedding به مدل وارد می‌گردد. یک بردار قابل یادگیری با عنوان [class] نیز به داده‌ها اضافه می‌شود که نمایانگر خروجی نهایی است. این مجموعه ورودی‌ها سپس به Transformer Encoder منتقل می‌شوند که از مکانیزم Multi-Head Attention برای یادگیری ارتباطات بین بخش‌ها و یک MLP برای پردازش ویژگی‌ها بهره می‌برد. در نهایت، بردار [class] به‌عنوان خروجی نهایی برای طبقه‌بندی تصویر استفاده می‌شود. در ادامه، مدل بارگذاری شده و پس از تنظیم تعداد کلاس‌ها به ۴، معماری آن نمایش داده شده است.



مدل ViT به‌طور کلی از چهار بخش اصلی تشکیل شده است:

الف. تقسیم تصویر به پچ‌ها (Patch Embedding)

ابتدا تصویر ورودی به اندازه‌های ثابت (مثلاً 16x16 پیکسل) تقسیم می‌شود. هر پچ از تصویر به یک وکتور یکپارچه (embedding) تبدیل می‌شود که به‌عنوان ورودی به مدل ترنسفورمر ارسال می‌شود. این مرحله مشابه عملیات تبدیل کلمات به وکتورهای عددی در مدل‌های NLP است.

ب. توکن‌های موقعیت (Positional Embeddings)

از آنجایی که ترانسفورمرها به‌طور طبیعی ترتیب داده‌ها را نمی‌فهمند، توکن‌های موقعیت به پچ‌های تبدیل‌شده اضافه می‌شود تا اطلاعات مکانی تصویر حفظ شود. این توکن‌ها به‌عنوان افزونه به وکتورهای پچ‌ها اضافه شده و اطلاعات مربوط به موقعیت مکانی پچ‌ها را به مدل منتقل می‌کنند.

ج. لایه‌های ترنسفورمر

ویژن ترنسفورمر شامل چندین لایه ترنسفورمر است که به‌طور مشابه به مدل‌های NLP عمل می‌کنند. این لایه‌ها شامل چندین لایه self-attention و feed-forward هستند که به مدل این امکان را می‌دهند که توجه خود را به بخش‌های مختلف تصویر جلب کرده و ویژگی‌های مختلف را استخراج کند.

د. لایه‌های خروجی (Head)

پس از عبور از لایه‌های ترنسفورمر، خروجی‌های مدل به یک لایه fully connected ارسال می‌شوند که وظیفه پیش‌بینی برچسب‌های کلاس یا ویژگی‌های مختلف تصویر را دارد. این لایه معمولاً به‌عنوان "classification head" شناخته می‌شود.

مزایای Vision Transformer

- **عملکرد بالا در داده‌های بزرگ**: ViT قادر است با داده‌های بسیار بزرگ، عملکرد بهتری نسبت به CNNها داشته باشد. این مدل به ویژه زمانی که تعداد داده‌های آموزش زیاد باشد، به‌طور قابل توجهی بهتر عمل می‌کند.
- **ساختار ساده‌تر**: با استفاده از ترانسفورمرها، مدل ViT ساختاری ساده و شفاف دارد که می‌تواند به‌راحتی برای مشکلات مختلف سفارشی‌سازی شود.
- **استفاده از ترانسفورمرها**: این مدل قادر است روابط پیچیده‌تر در داده‌های تصویری را مدل‌سازی کند، همانطور که در NLP موفق بوده است.

معایب Vision Transformer

- **نیاز به داده‌های زیاد**: یکی از معایب اصلی ViT این است که برای عملکرد بهینه نیاز به مجموعه داده‌های بسیار بزرگ دارد. مدل‌های CNN معمولاً قادر به آموزش مؤثرتر با داده‌های کمتر هستند.
- **محاسبات سنگین**: به دلیل استفاده از لایه‌های ترنسفورمر که پیچیدگی محاسباتی بالایی دارند، ViT ممکن است نیاز به منابع محاسباتی زیادی داشته باشد، به ویژه در مقیاس‌های بزرگ.
- **زمان آموزش طولانی**: به‌دلیل پیچیدگی مدل و نیاز به داده‌های بزرگ، زمان آموزش این مدل معمولاً بیشتر از مدل‌های CNN است.

در تمامی مدل‌ها، از روش early stopping استفاده شده که مقدار patience آن برابر 18 در نظر گرفته شده است، بدین معنی که بعد از اولین نقطه‌ای که برای early stopping تصمیم گرفته می‌شود، به تعداد 18 ایپاک به مدل اجازه آموزش داده می‌شود تا خود را از پروسه‌ی early stopping خارج کند، وگرنه فرآیند آموزش پایان می‌یابد. توجه شود در نتیجه این عمل مدل بهترین پارامترهای خود را ذخیره کرده است.

مقدار learning rate ای که با بهینه‌ساز Adam داده می‌شود، برابر 3×10^{-4} است. این مقداری رایج به عنوان ورودی lr می‌باشد.

به صورت پیش‌فرض مقدار 100 ایپاک در نظر گرفته شده است که همه‌ی مدل‌ها فرایند آموزش را در تعداد ایپاک کمتری تمام کرده‌اند.

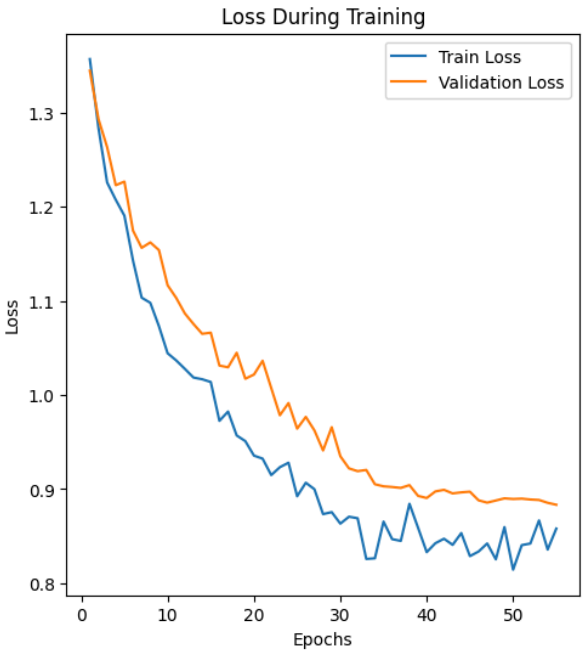
به مدل‌ها مقدار scheduler patience نیز نظیر شده که وظیفه تنظیم نرخ یادگیری را بر عهده دارد. اگر مقدار scheduler patience برابر -1 نباشد، هر چند وقت یک‌بار نرخ یادگیری نصف میشود. این اتفاق زمانی می‌افتد که early_stop_counter به طور دقیق مضربی از scheduler_patience باشد. در این حالت نرخ یادگیری نصف مقدار قبلی شده و تغییرات نرخ یادگیری نیز چاپ میشود.

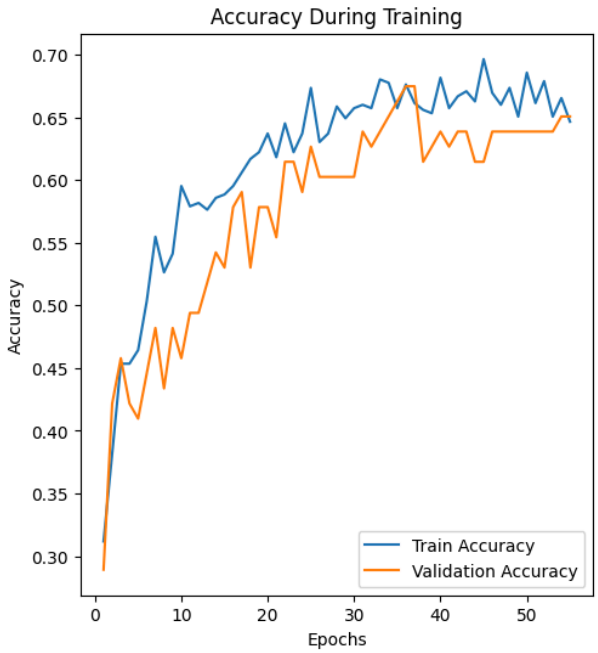
2-3-2. آموزش مدل ViT با حالت اول

در این حالت، فقط دسته‌بند Classifier قابل آموزش می‌باشد.

Total Parameters Count	Trainable Parameters
85,801,732	3,076

2-3-3. بررسی نتایج مدل ViT با حالت اول





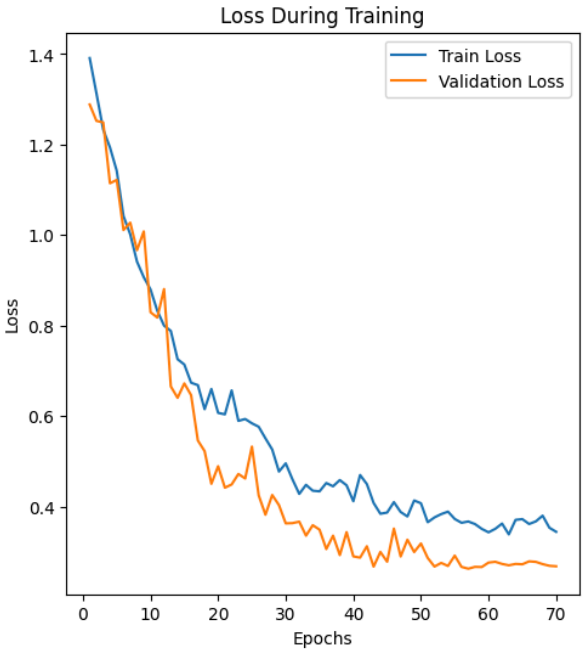
2-3-4. آموزش مدل ViT با حالت دوم

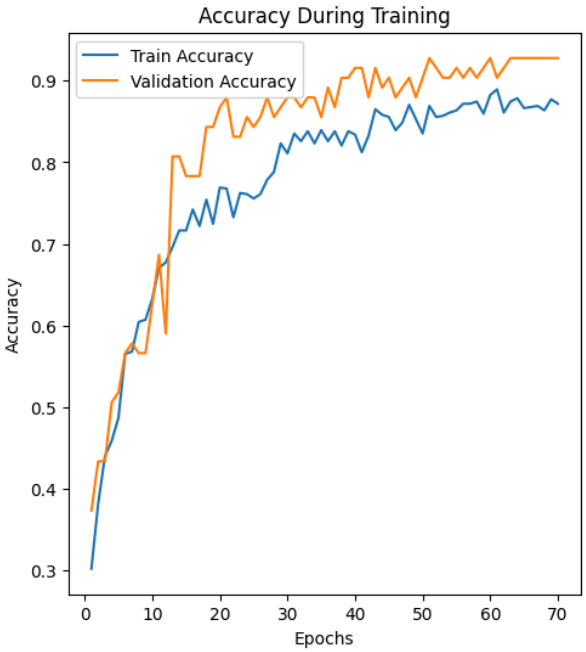
در این حالت، دو لایه‌ی اول Encoder قابل آموزش می‌باشد.

Total Parameters Count	Trainable Parameters
85,801,732	14,175,744

از آنجایی که در این حالت تنها دو لایه‌ی اول encoder را آموزش می‌دهیم، انتظار

2-3-5. بررسی نتایج مدل ViT با حالت دوم



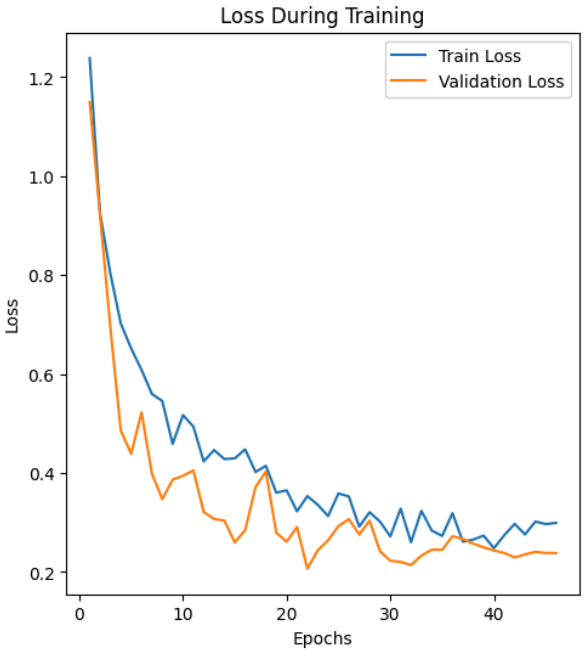


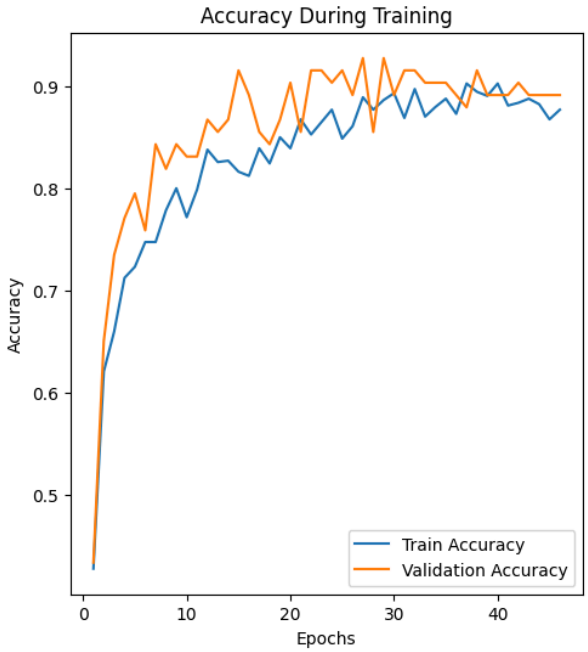
2-3-6. آموزش مدل ViT با حالت سوم

در این حالت، دو لایه‌ی آخر Encoder قابل آموزش می‌باشد.

Total Parameters Count	Trainable Parameters
85,801,732	14,175,744

2-3-7. بررسی نتایج مدل ViT با حالت سوم





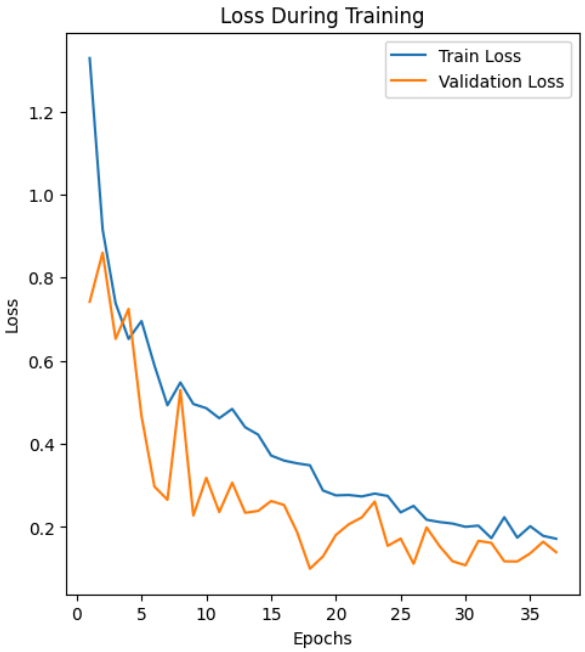
2-3-8. آموزش مدل ViT با حالت چهارم

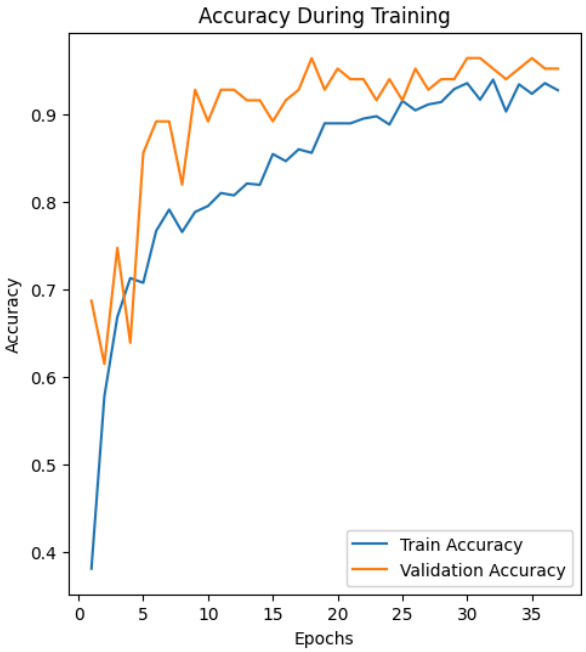
در این حالت، تمامی لایه‌ها قابل آموزش می‌باشد.

Total Parameters Count	Trainable Parameters
85,801,732	85,801,732

همانطور که انتظار می‌رفت، در این حالت تعداد پارامترهای قابل آموزش از بقیه موارد بزرگتر می‌باشد.

2-3-9. بررسی نتایج مدل ViT با حالت چهارم





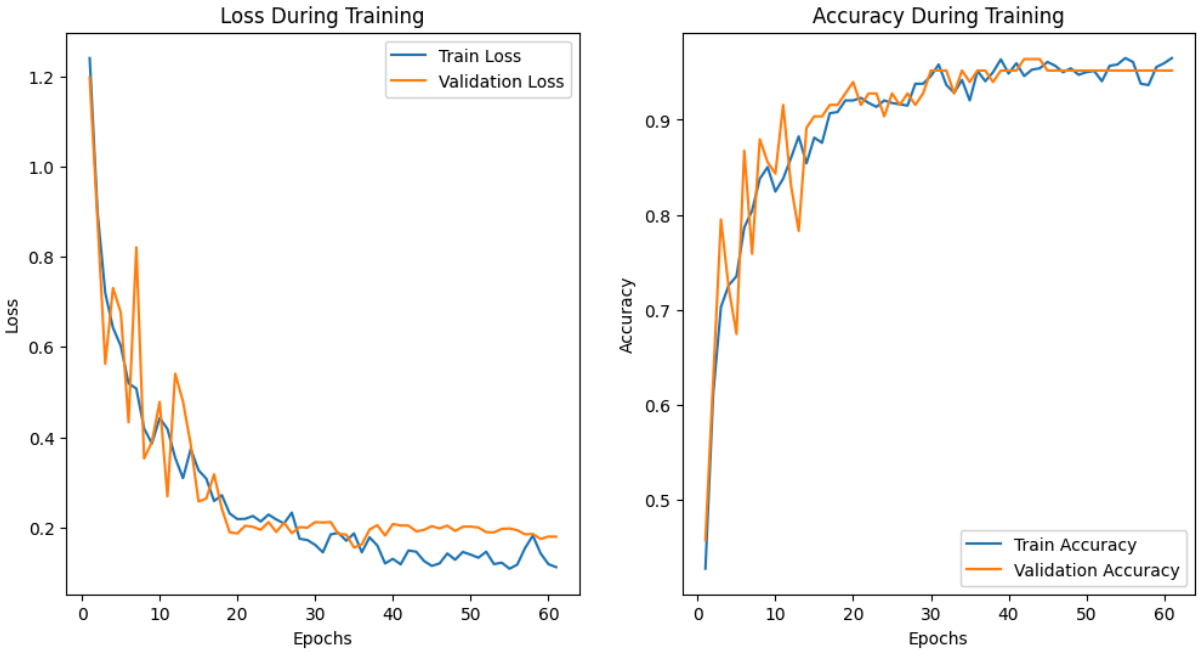
2-3-10. آموزش مدل CNN به طور کامل آموزش دیده

در این حالت، تمامی لایه‌ها قابل آموزش می‌باشد.

Total Parameters Count	Trainable Parameters
6,957,956	6,957,956

همانطور که انتظار می‌رفت، تمامی پارامترها قابل آموزش هستند.

2-3-11. بررسی نتایج مدل CNN به طور کامل آموزش دیده



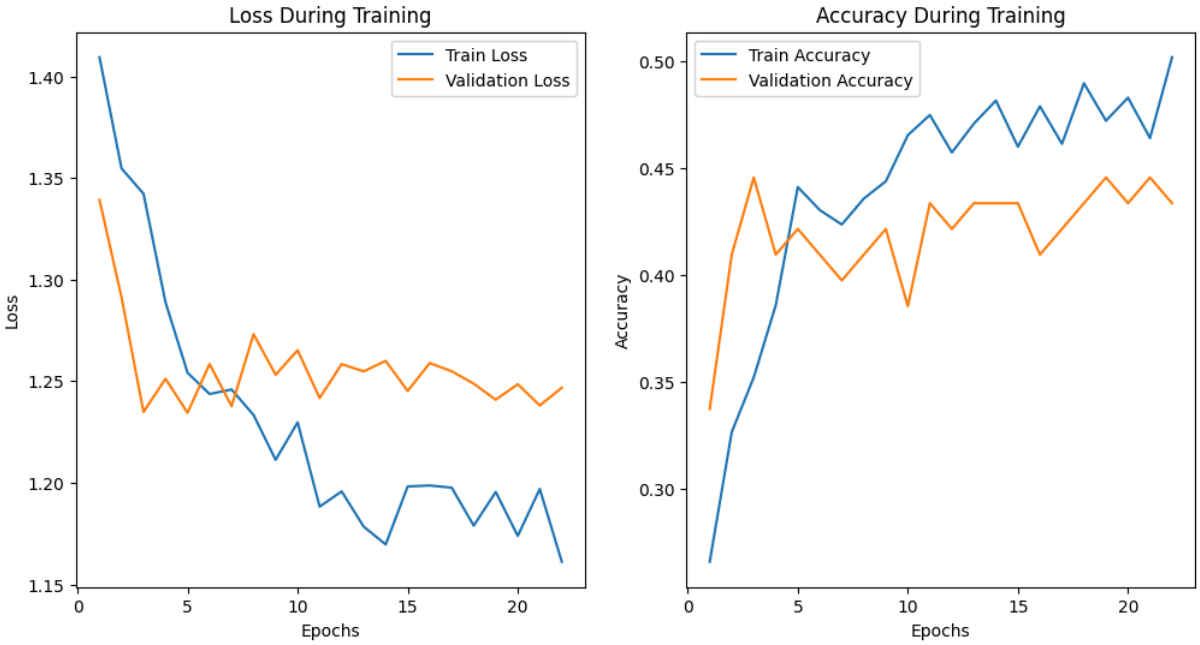
2-3-12. آموزش مدل CNN با فقط لایه‌ی Classifier آموزش دیده

در این حالت، تنها لایه‌ی Classifier قابل آموزش می‌باشد.

Total Parameters Count	Trainable Parameters
6,957,956	4,100

همانطور که مشاهده می‌شود، تعداد پارامترهای قابل آموزش مدل کاهش یافته است.

2-3-13. بررسی نتایج مدل CNN با فقط لایه‌ی Classifier آموزش دیده



2-4. تحلیل و نتیجه‌گیری

2-4-1. مقایسه مدل‌های آموزش دیده روی تمامی لایه‌ها

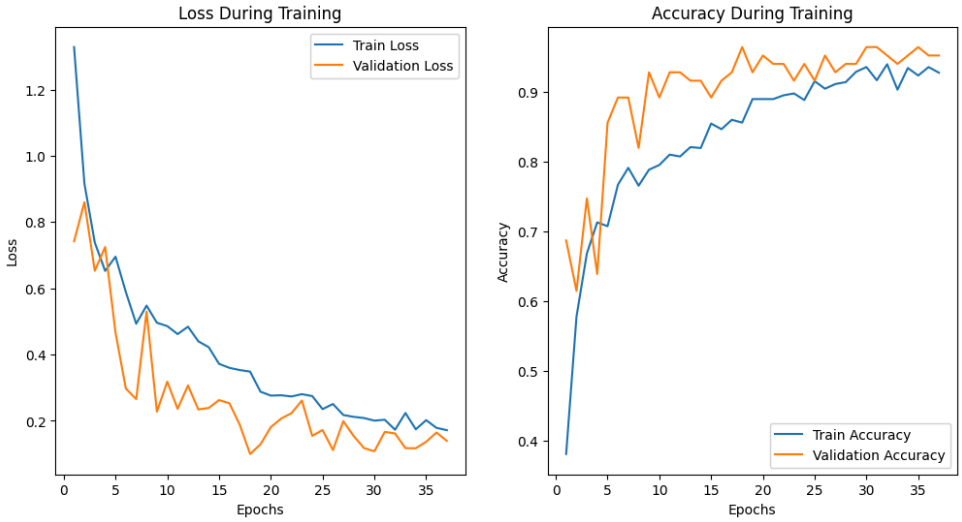
مدل ViT:

Total Parameters Count	Trainable Parameters
85,801,732	85,801,732

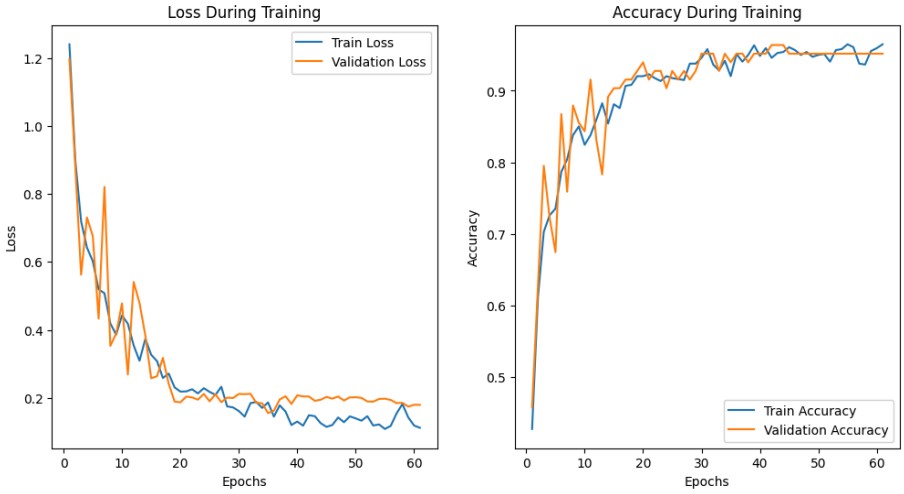
مدل CNN:

Total Parameters Count	Trainable Parameters
6,957,956	6,957,956

مدل ViT ذاتا تعداد پارامترهای قابل آموزش خیلی بزرگ تر از CNN می‌باشد.



شکل . **نتایج مدل ViT**



شکل . **نتایج مدل CNN**

همانطور که از نمودارها قابل مشاهده است، early stopping در مدل ViT زودتر از CNN فراخوانی شده، به همین دلیل مدل اول در 40 ایپاک و مدل دوم در 60 ایپک آموزش دیده است. علت این مورد تصادفی بودن است.

با مقایسه این دو مدل، متوجه می‌شویم مدل اول تغییرات Smooth تری نسبت به مدل دوم ارائه داده است، که علت آن، پیچیدگی بیشتر مدل گوگل می‌باشد. با صرف نظر از Smoothness دو مدل و اینکه در این مورد عملکرد نهایی دو مدل تقریبا یکی شده برای مقایسه دقیق‌تر و تفاوت عملکرد دو مدل، به سراغ بخش بعد می‌رویم.

2-4-2. مقایسه مدل‌های آموزش دیده روی لایه Classifier

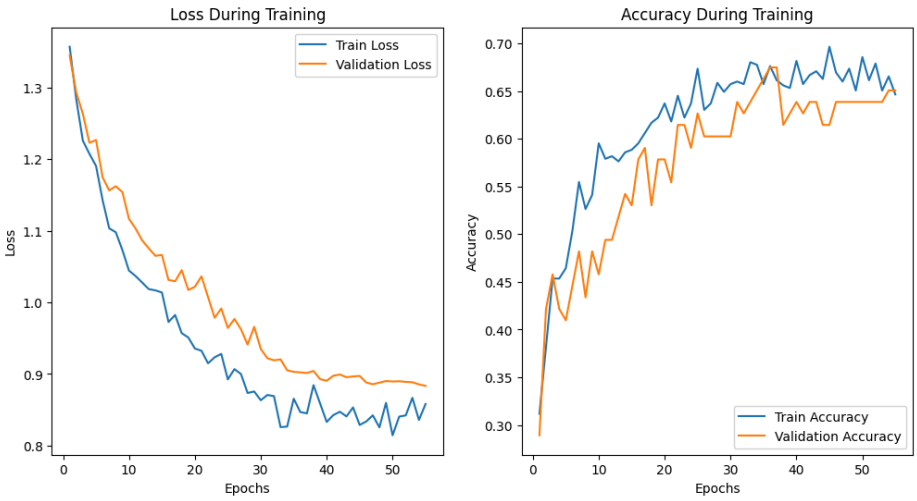
مدل ViT:

Total Parameters Count	Trainable Parameters
85,801,732	3,076

مدل CNN:

Total Parameters Count	Trainable Parameters
6,957,956	4,100

در مدل CNN، تعداد پارامترهای قابل آموزش بیشتری قرار دارد. البته در مدل ViT ذاتا تعداد پارامترهای قابل آموزش خیلی بزرگ تر از CNN می‌باشد.



شکل . **نتایج مدل ViT**

شکل . **نتایج مدل CNN**

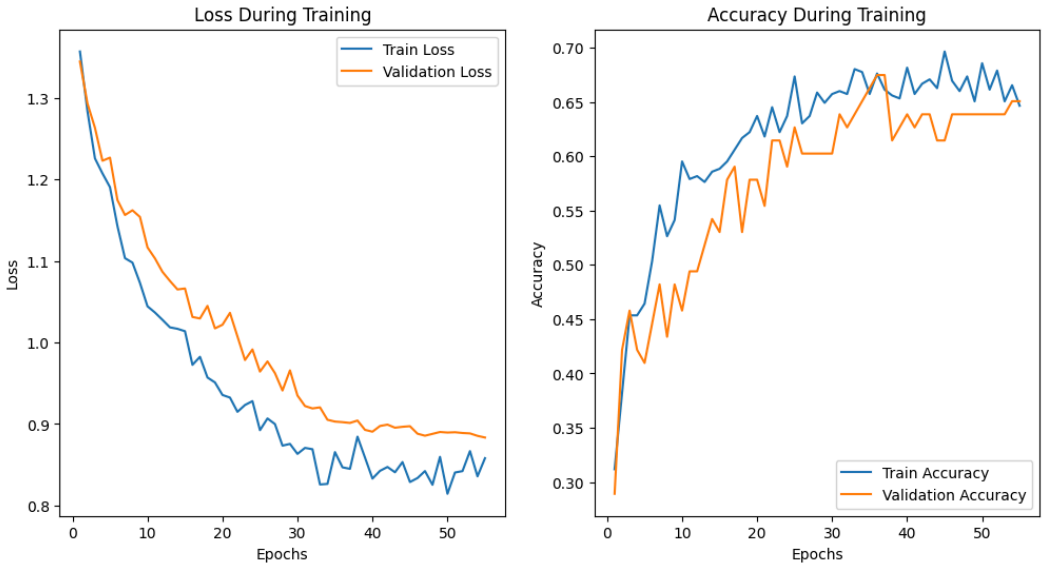
همانطور که انتظار می‌رفت، مدل با تعداد پارامترهای آموزش دیده بیشتر، نتیجه بهتری داده است. همینطور قابل مشاهده است که مدل CNN نسبت به مدل گوگلی دچار اورفیت بیشتری شده است.

علت این موضوع، قوی بودن مدل گوگل روی این دست مجموعه‌های داده و مکانیزم قوی تر آن می‌باشد که با تنها آموزش داده لایه Classifier آن، به نتیجه بهتری رسیده‌ایم.

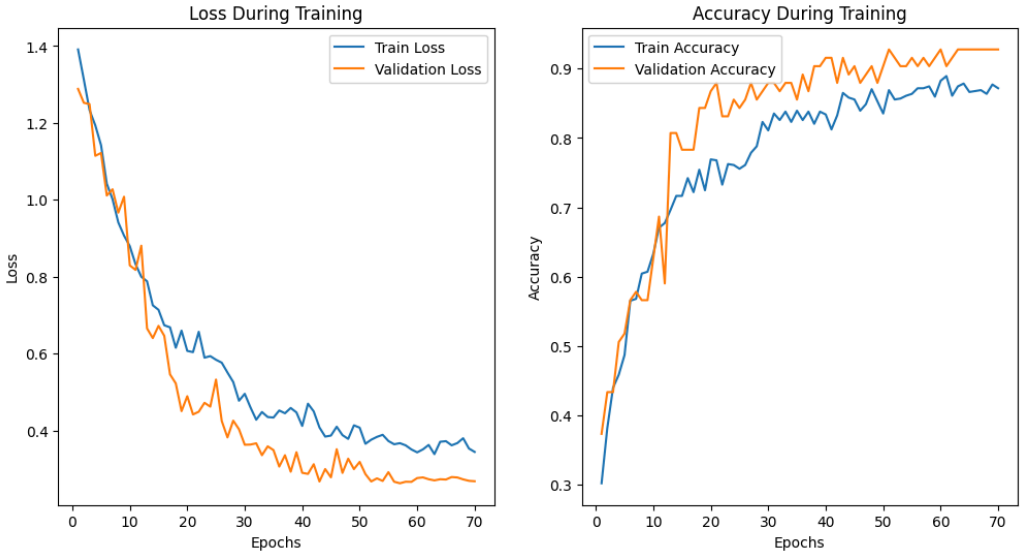
2-4-3. مقایسه مدل‌های ViT در حالت‌های مختلف

در این بخش به ترتیب تعداد پارامترها به ترتیب کوچک به بزرگ بیان شده است. توجه کنید مدل در حالت دوم و سوم تعداد پارامتر قابل آموزش برابری دارد.

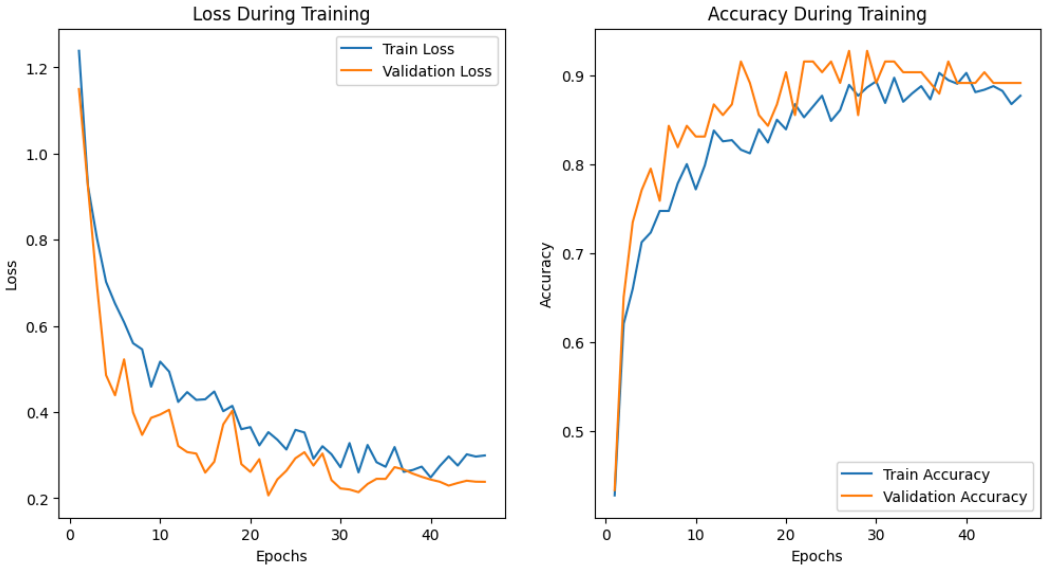
مقدار loss و نحوه کاهش آن در این چهار حالت، مقدار معقولی بوده و با دقت مدل همخوانی دارد.



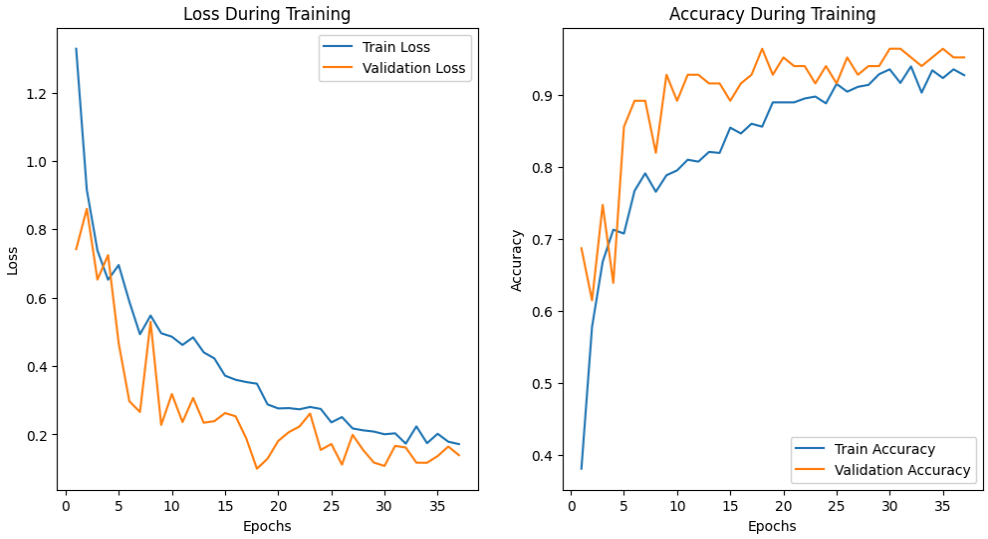
شکل . **نتایج مدل ViT در حالت اول**



شکل . **نتایج مدل ViT در حالت دوم**



شکل . **نتایج مدل ViT در حالت سوم**



شکل . **نتایج مدل ViT در حالت چهارم**

- مدل در حالت **اول،** کمترین انعطاف‌پذیری را دارد زیرا تنها لایه دسته‌بند روی داده‌های جدید تنظیم می‌شود. عملکرد مدل به شدت وابسته به ویژگی‌های استخراج شده توسط وزن‌های از پیش آموزش‌داده‌شده است. در این حالت، مدل نمی‌تواند ویژگی‌های سطح پایین یا میانی را برای داده‌های جدید تطبیق دهد. این محدودیت باعث می‌شود دقت کمتری نسبت به

مدل‌هایی که لایه‌های بیشتری قابل آموزش هستند داشته باشد، به‌خصوص در صورتی که داده‌های جدید با داده‌های استفاده‌شده در پیش‌آموزش تفاوت زیادی داشته باشند.

- مدل در حالت **دوم** توانایی تنظیم ویژگی‌های سطح بالا را دارد و بنابراین می‌تواند تا حدی با داده‌های جدید سازگار شود. عملکرد آن بهتر از مدل اول است، اما همچنان محدودیت‌هایی در یادگیری ویژگی‌های سطح پایین‌تر دارد. دو لایه آخر در Encoder معمولاً مسئول استخراج ویژگی‌های سطح بالا هستند. قابل آموزش بودن این لایه‌ها باعث می‌شود مدل بتواند ویژگی‌های پیچیده‌تری را یاد بگیرد و با داده‌های جدید بهتر سازگار شود.
- مدل در حالت **سوم،** به بهبود یادگیری ویژگی‌های سطح پایین کمک می‌کند، اما ویژگی‌های سطح بالا همچنان ثابت باقی می‌مانند. در برخی موارد، این مدل ممکن است عملکرد ضعیف‌تری نسبت به مدل دوم داشته باشد، زیرا عدم تغییر در لایه‌های بالاتر می‌تواند سازگاری کلی را محدود کند. لایه‌های اول وظیفه استخراج ویژگی‌های پایه را بر عهده دارند. قابل آموزش بودن این لایه‌ها به مدل اجازه می‌دهد اطلاعات اولیه تصویر را با داده‌های جدید تطبیق دهد.
- مدل در حالت **چهارم،** بالاترین انعطاف‌پذیری را دارد و می‌تواند به طور کامل برای داده‌های جدید تنظیم شود. در نتیجه، بهترین عملکرد را در میان چهار مدل ارائه می‌دهد. قابل آموزش بودن تمامی لایه‌ها به مدل اجازه می‌دهد ویژگی‌های سطح پایین، میانی و بالا را همزمان با داده‌های جدید بهینه کند. این انعطاف‌پذیری منجر به یادگیری بهتر و عملکرد بالاتر می‌شود، به خصوص در صورتی که داده‌های جدید تفاوت زیادی با داده‌های پیش‌آموزش داشته باشند.

علت بهتر بودن نتایج روی داده تست در برخی مدل‌ها این است که Augmentation روی داده‌ها unsample است. این Augmentation ها روی داده تست پیاده‌سازی نمی‌شوند و تنها روی دیتای ترین اعمال می‌شود تا مدل به صورت General آموزش ببیند.

هیچ کدام از مدل‌ها به غیر از مدل CNN با فقط لایه‌ی Classifier آموزش دیده، هم Overfit نکرده‌اند، چرا که مقدار Train به صورت عجیب و غریب بهتر از Test نمی‌باشد. علت Overfit شدن آن مدل هم کم بودن تعداد پارامترهای آموزش دیده شده و ساده بودن مدل نسبت به مدل گوگل می‌باشد.

1. کدام مدل در مجموعه داده شما عملکرد بهتری داشت؟

به طور کلی حالت چهارم از ViT عملکرد بهتر، Smooth تری روی مجموعه داده‌ی تست نشان داده‌است. علت این مورد و مقایسه در زیر بخش 2 از همین بخش توضیح داده شده است.

2. آیا ViT در شرایط موجود (مثلا داده‌های نویز دار یا کم‌حجم) توانسته جایگزین مناسبی برای CNN باشد؟

مدل ViT به دلیل استفاده از مکانیزم Self-Attention، توانایی یادگیری روابط کلی و شناسایی الگوهای پیچیده را دارد. در نتیجه این معماری به جایگزینی مناسب برای CNN تبدیل شود. این مورد مخصوصا زمانی قابل مشاهده است که داده‌ها کم یا دارای نویز باشند. این تفاوت روی نمودارهای ارائه شده در زیربخش شماره 1 از این بخش توضیح داده شده است. در مقایسه با ViT، DenseNet-121 حتی با تنظیم محدود (Fine-Tuning) نیز عملکرد بهتری داشته است. ممکن است با تغییرات در تنظیمات هایپر پارامترها، DenseNet-121 نتایج بهتری هم ارائه دهد. بنابراین، اگر منابع کافی در دسترس باشد، ViT گزینه‌ای قدرتمند است، اما در شرایط محدودیت منابع یا نیاز به سرعت پردازش بیشتر، استفاده از CNN همچنان توصیه می‌شود.