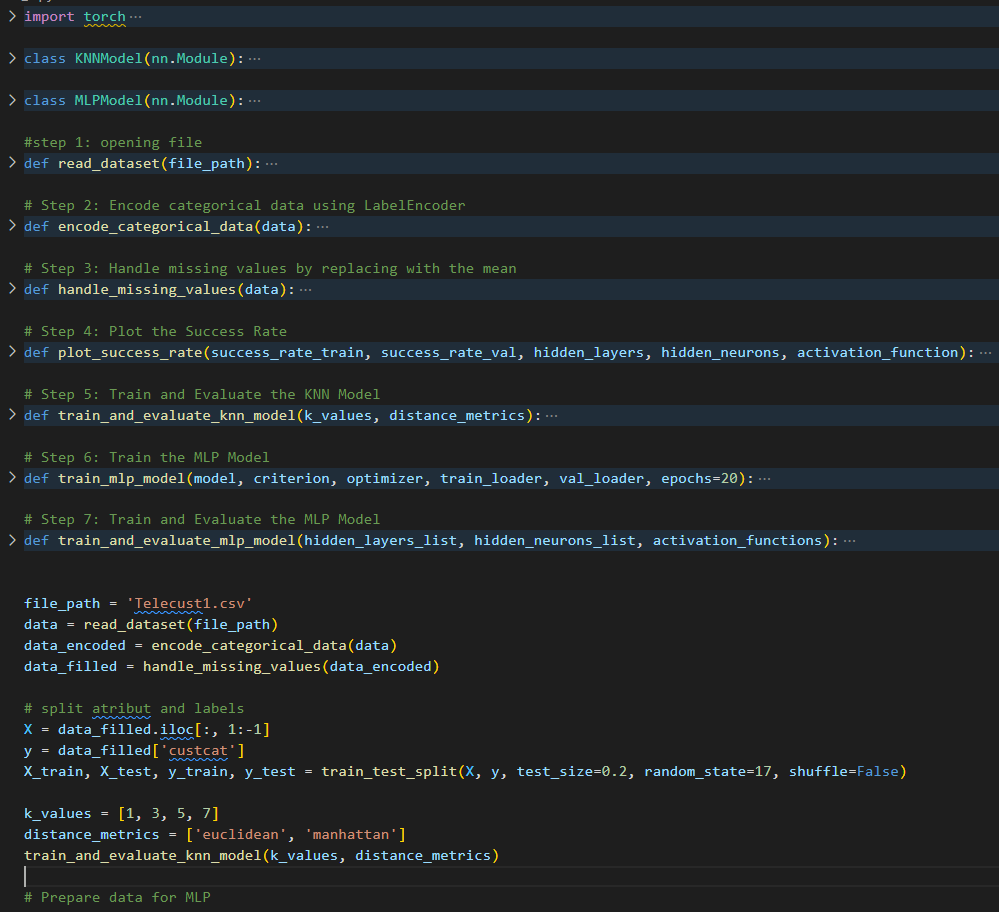
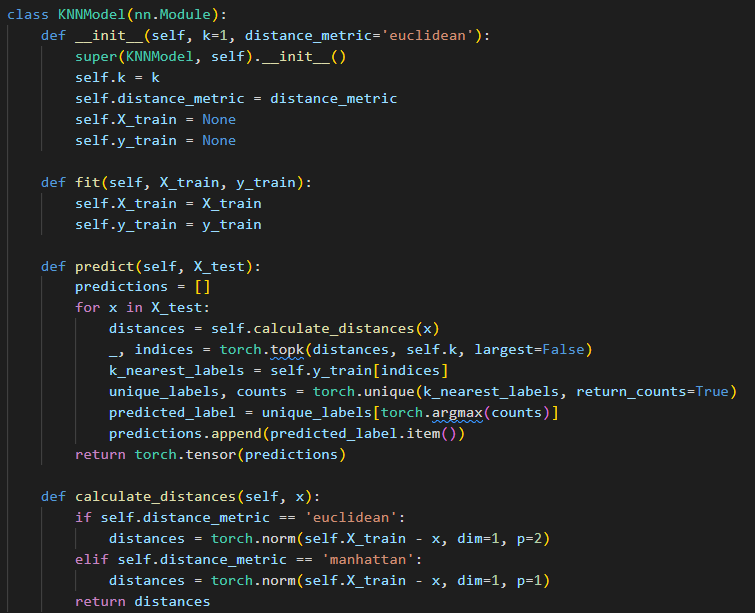
علی آرمان 4023624002

در پیاده سازی ابتدا یک بار کد ها نوشته شد و در نهایت مرتب شد و به صورت بهینه کنار هم قرار گرفت.

در ابتدا نگارش کد ها گزارش شده و بعد از آن اجرای کد و در هر قسمت با رنگ قرمز شماره گزارش مشخص شده.

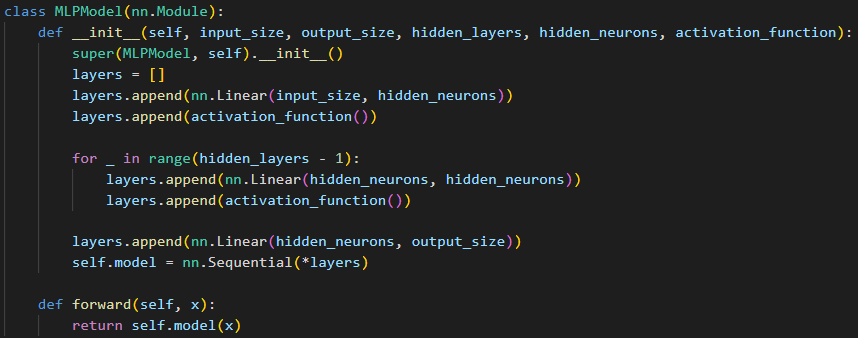


در کلاس KNNModel ما طبق اموزش tourch تابع سازنده، تابع فیت و تابع پریدیکت داریم:



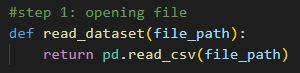
تابع calculate\_distances برای مشخص کردن نوع فاصله هست

در کلاس MLPModel مشابه کلاس قبلی طبق آموزش پیاده سازی شده :

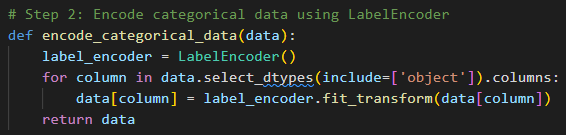


در تابع سازنده، سایز ورودی و خروجی، تعداد لایه ها و نورون ها و همچنین تابع فعال ساز را مشخص کردیم.

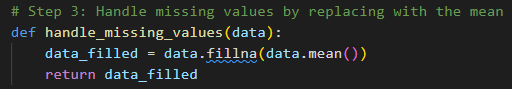
در تابع read\_dataset فایل را باز میکنیم:



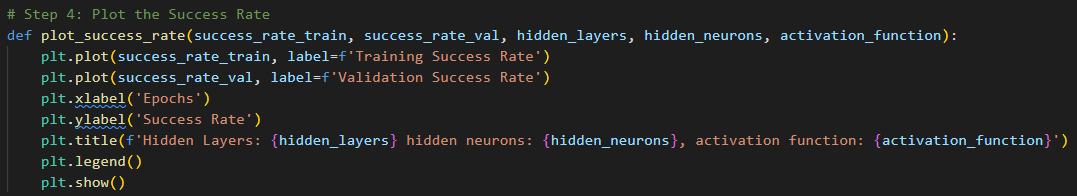
در تابع encode\_categorical\_data ما داده را انکود میکنیم تا قابل پردازش کنیم:



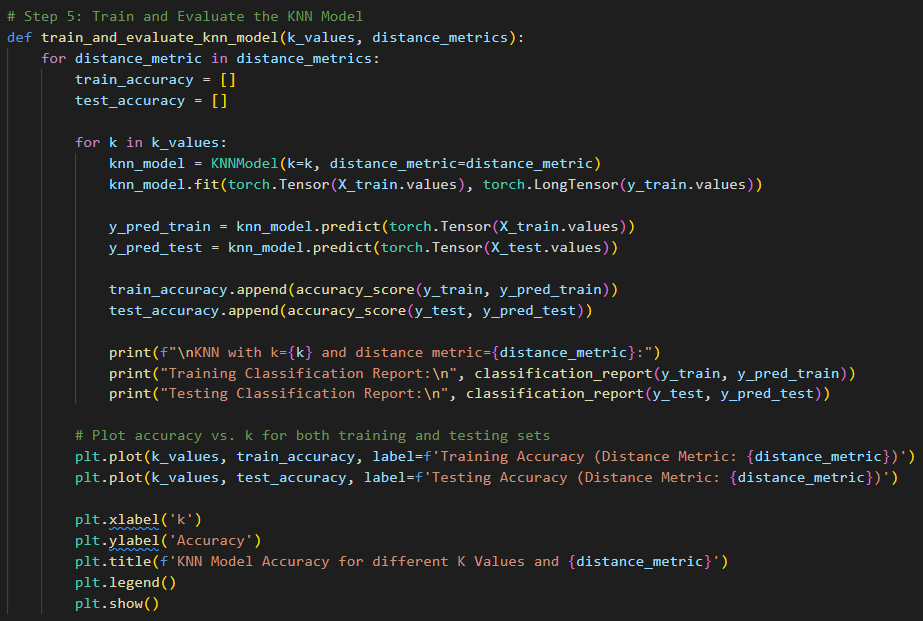
در تابع handle\_missing\_values مقادیری که null هستند را با میانگین پر میکنیم:



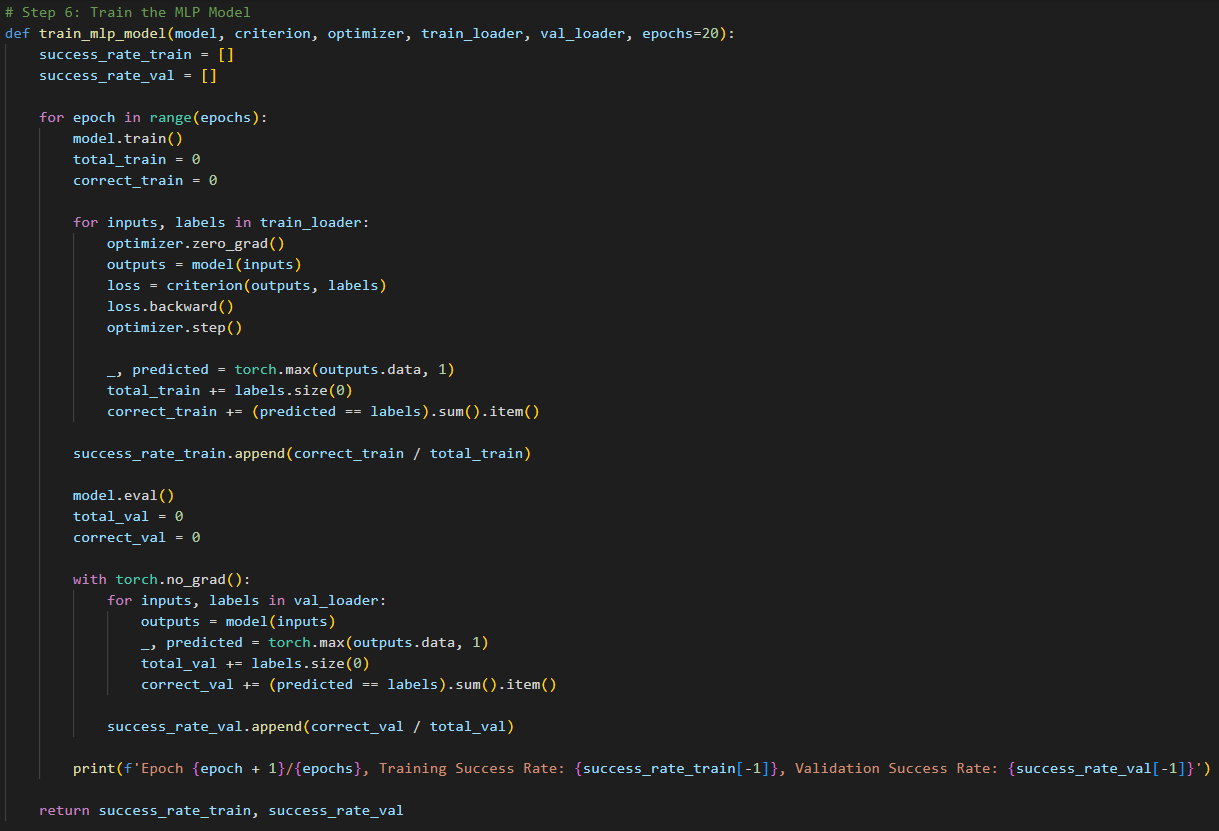
در تابع plot\_success\_rate ما خروجی شبکه عصبی را با نمودار نشان میدهیم. که تعداد لایه و نورون و تابع فعال ساز در تایتل آن مشخص است.



در تابع train\_and\_evaluate\_knn\_model مدل knn خود را با ترکیب تمام حالات نوع فاصله و تعداد k آموزش دادیم و در نهایت با پلات خروجی را برای دو نوع فاصله نمایش دادیم.



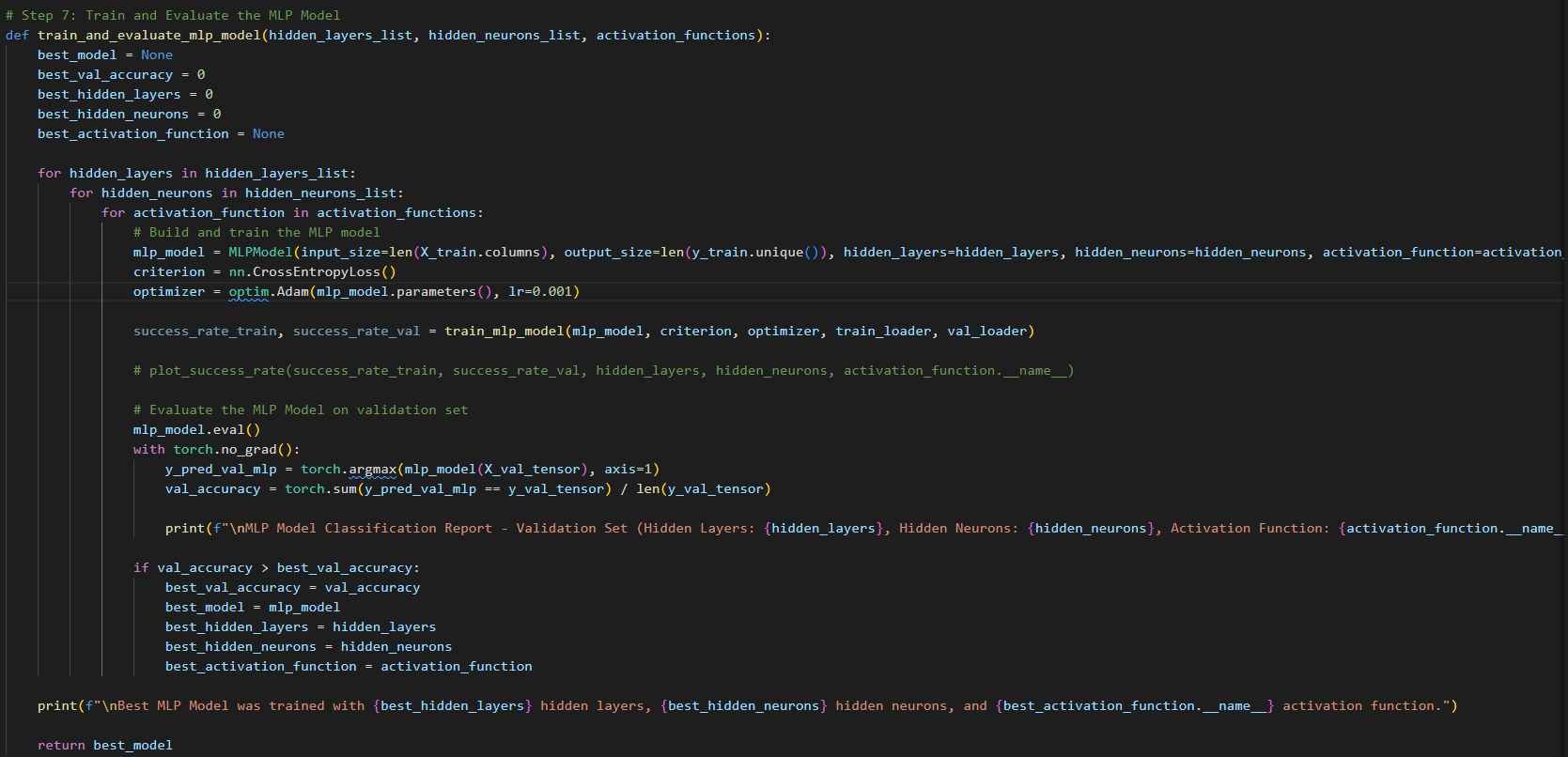
در تابع train\_mlp\_model طبق آموزش های موجود به تعداد دور اموزش(epochs) مدل را آموزش میدهیم، در هر بار مقدار خطا را حساب میکنیم و به مقدار های قبلی لیست اضافه میکنیم. و در نهایت خروجی را پرینت میکنیم:



در تابع train\_and\_evaluate\_mlp\_model ما شبکه را با تعداد لایه و تعداد نورون و توابع فعال ساز مختلف در تمام حالت ها آموزش میدهیم و در هر بار بررسی میکنیم اگر جواب به دست آمده بهتر باشد با جواب بهتر قبلی جایگزین میکنیم.

همچنین از plot\_success\_rate برای نمایش نتیجه این آموزش استفاده شده و در نهایت بهترین حالت در خروجی چاپ شده.

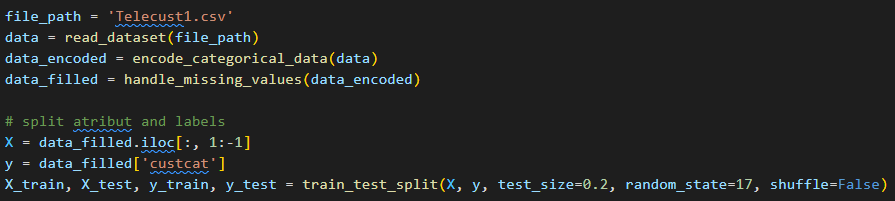
دلیل اینکه تمام حالت های ممکن برای تعداد لایه و تعداد نورون و توابع فعال ساز مختلف باهم تست شده این بوده که ممکن است از بین تعداد لایه 5 بهترین حالت باشد(عدد 5 فرضی است) ولی با تغییر تعداد نورون با 2 لایه جواب بهتری برسیم، این ترکیب برای تابع فعال ساز هم می باشد.



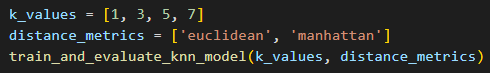
قسمت های اجرایی کد:

ابتدا نام فایل مشخص شده و فایل خوانده شده و انکود کردن و هندل کردن داده های null صورت گرفته.

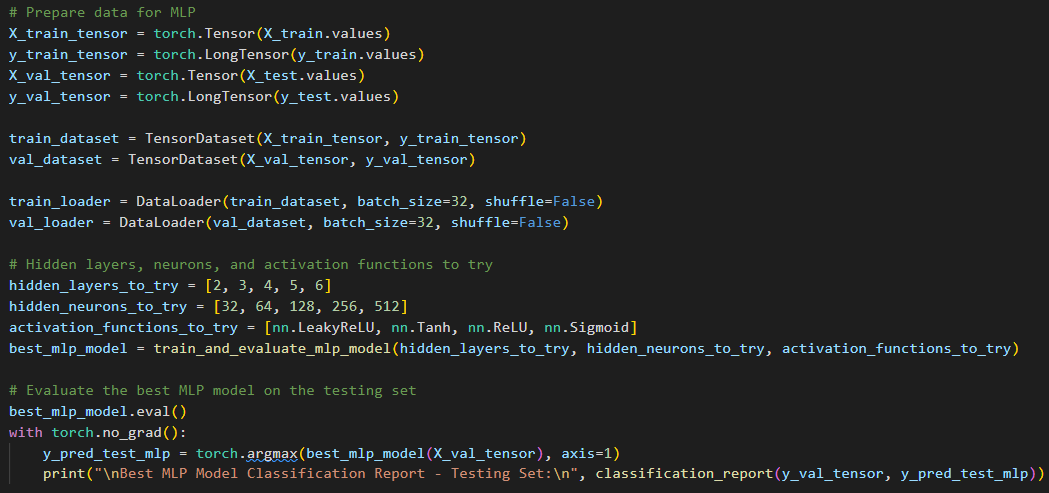
سپس ویژگی ها و دسته مشتریان از هم جدا شده اند و با train\_test\_split داده های اموزشی و تست تخصیص داده شده اند.



در ادامه تابع train\_and\_evaluate\_knn\_model با مقدار k های مختلف 1، 3، 5 و 7 و همچنین فاصله های Euclidean و Manhattan فراخوانی شده.



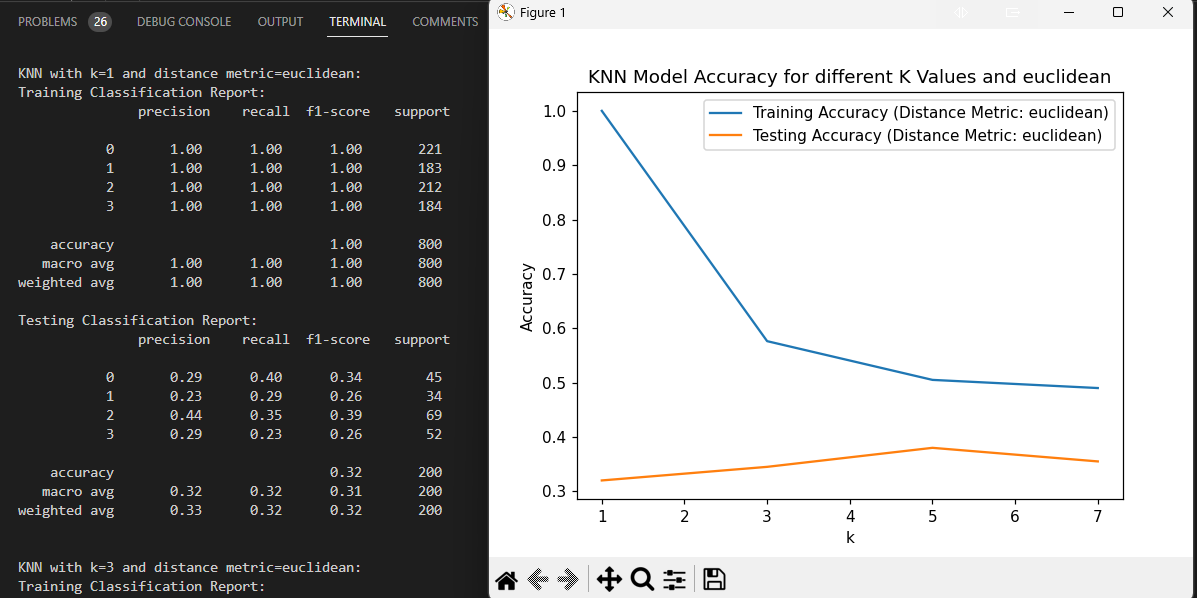
در ادامه داده ها با استفاده از tourch برای استفاده در شبکه عصبی آماده شده اند تابع train\_and\_evaluate\_mlp\_model با تعداد لایه 2 تا 6 و تعداد نورون 32 تا 512 و 4 تابع فعال ساز مختلف فراخونی شد و در نهایت گزارش بهترین کلاس چاپ شده:



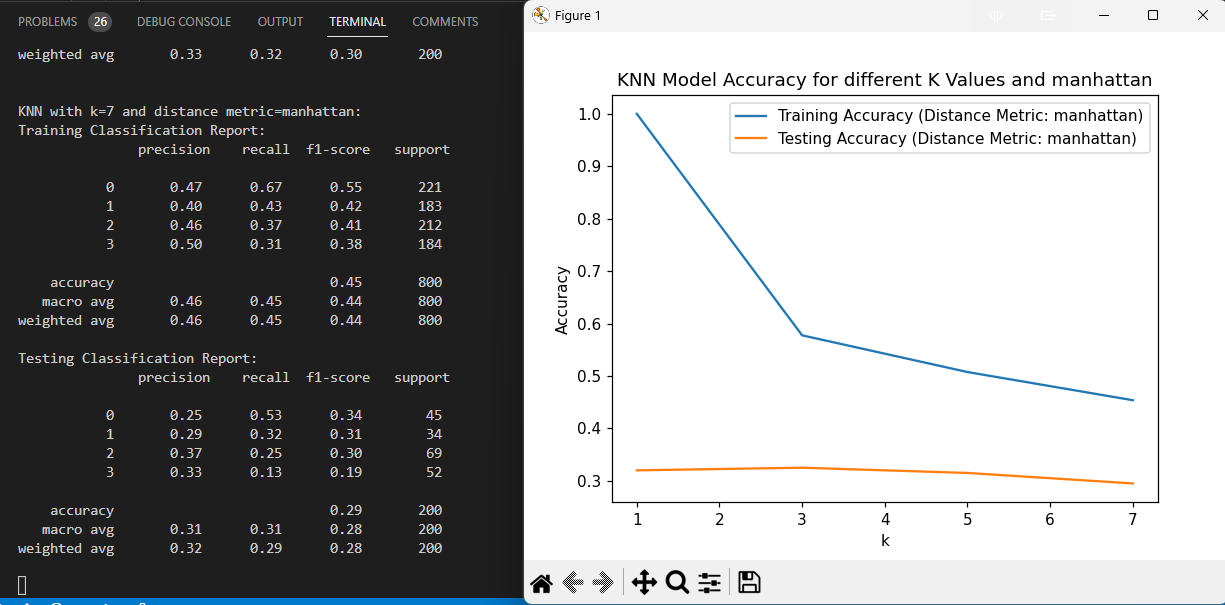
اجرای کد:

**گزارش 1**

نمایش نمودار و classification\_report مربوط به اجرا knn با k های مختلف و Euclidean:



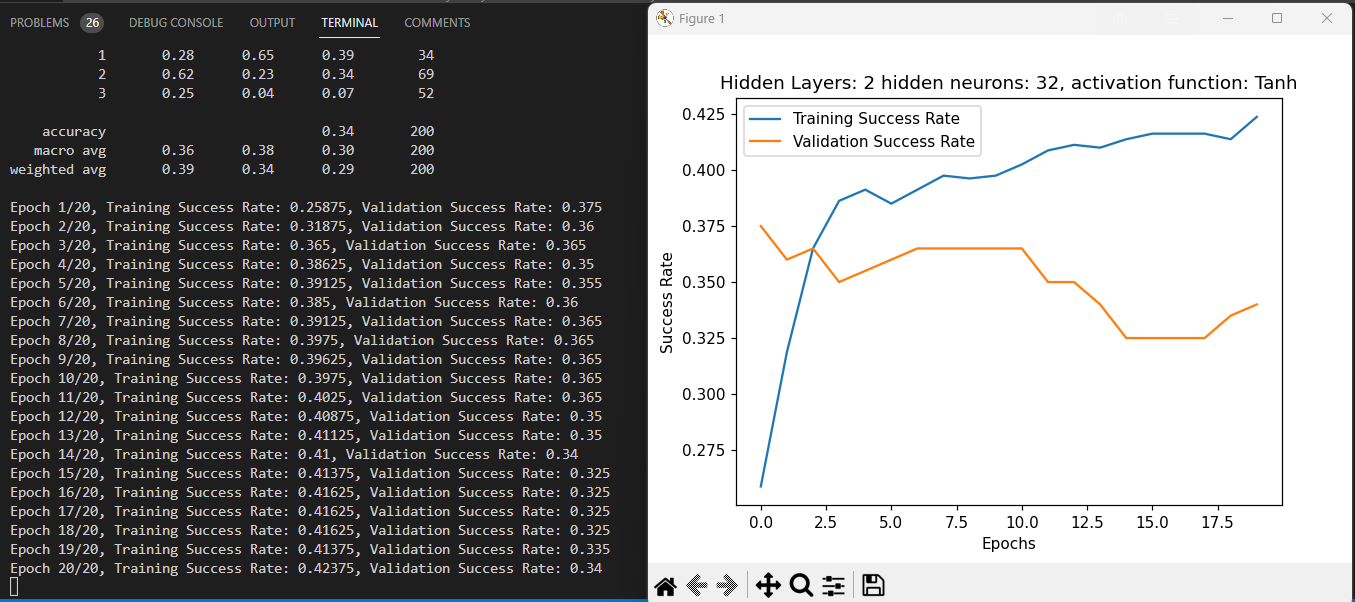
نمایش نمودار و classification\_report مربوط به اجرا knn با k های مختلف و manhattan:

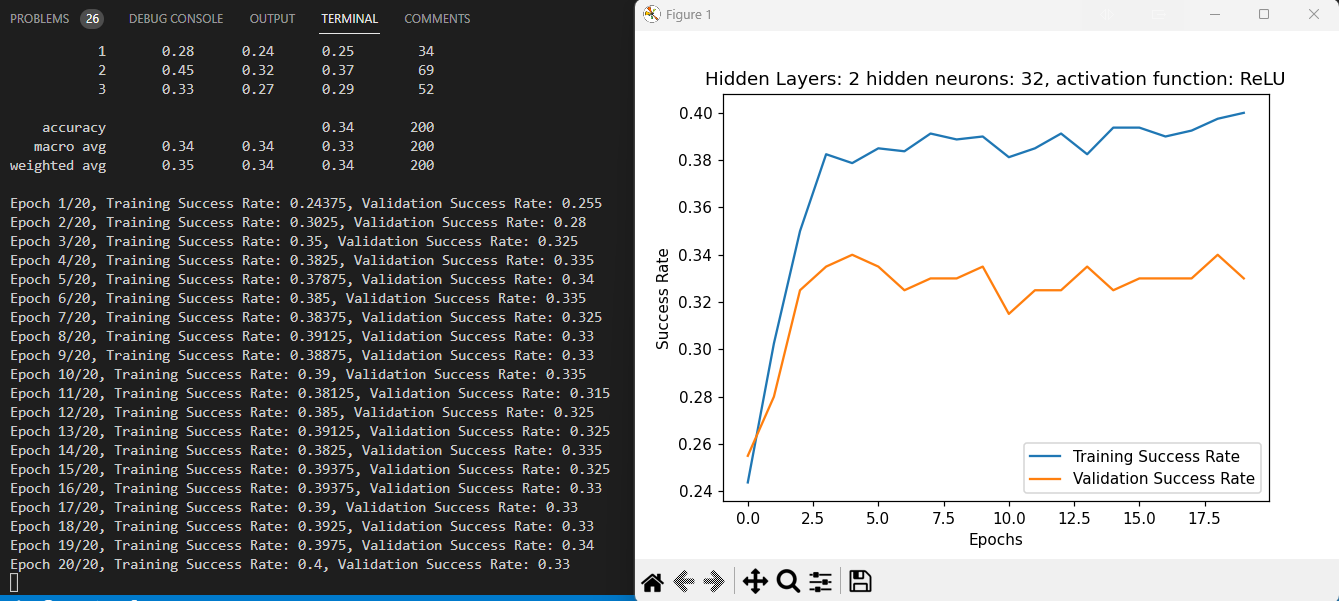


همان طور که پیداست فاصله منهتن در شروع کار در تست بهتر بوده ولی فاصله اقلیدسی در نهایت در تست بهتر بوده. و هر دو در ترین تقریبا مشابه هم دیگر بوده اند. و همان طور که از نتایج پیداست دقت فاصله اقلیدسی بهتر بوده.

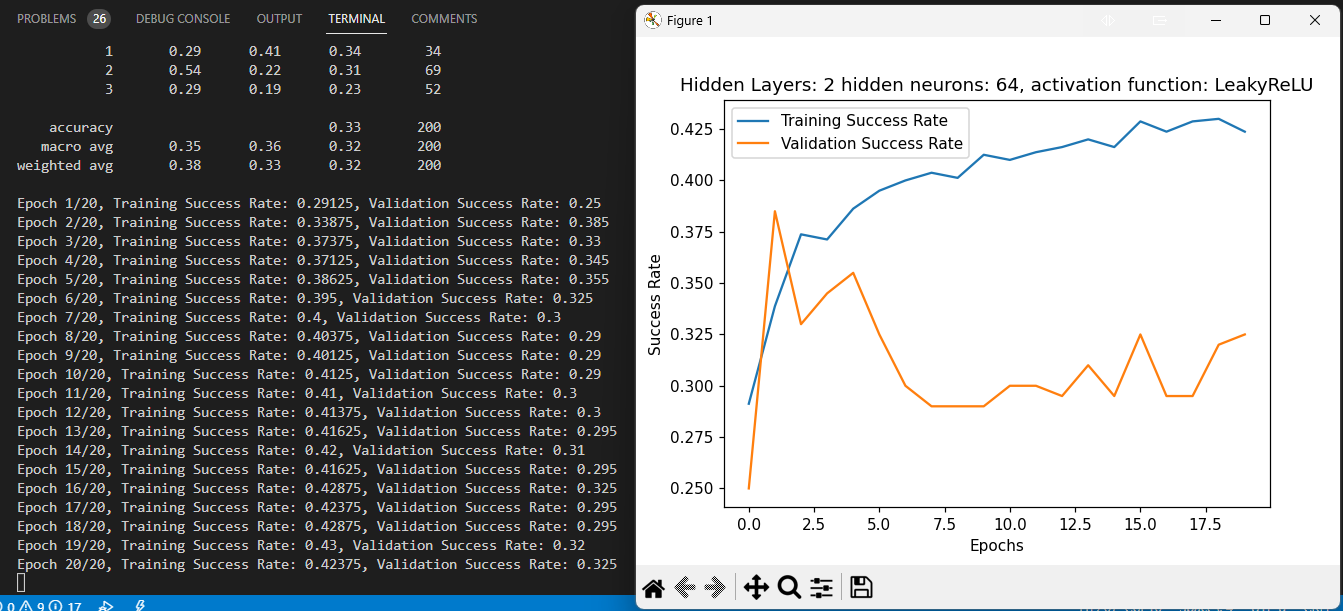
**گزارش 2 و 3 و 4 و 5**

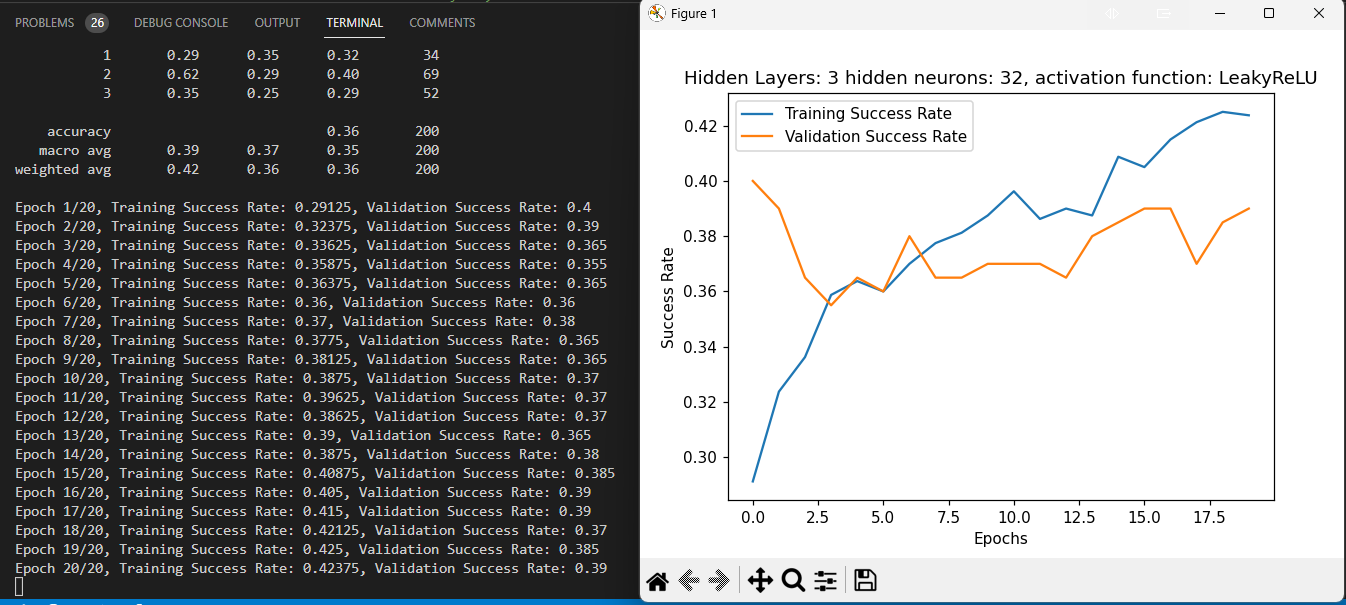
نمایش نمودار و classification\_report مربوط به اجرا mlp با تعداد لایه و تعداد نورون و توابع فعال ساز مختلف انجام شده(تعداد کل حالت ها زیاد می باشد اسکرین اجرا چند حالت در ادامه اورده شده):



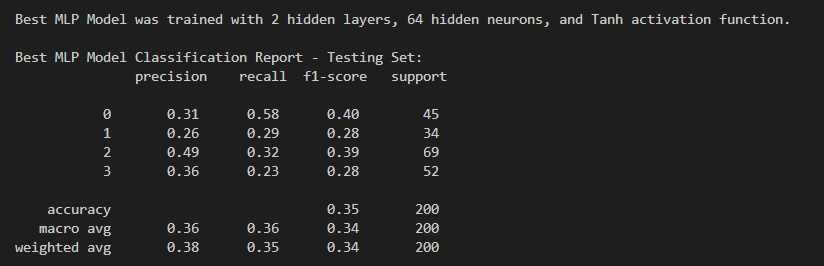




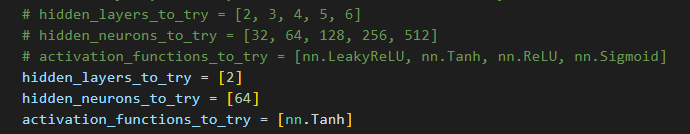


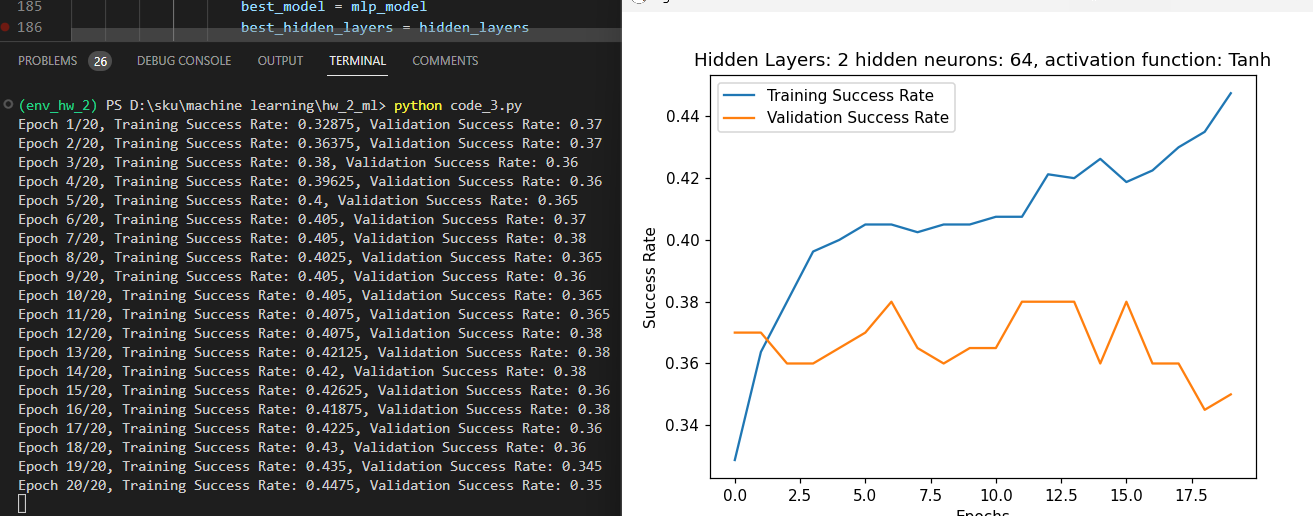


که بهترین جواب طبق اجرا کد به صورت زیر می باشد:

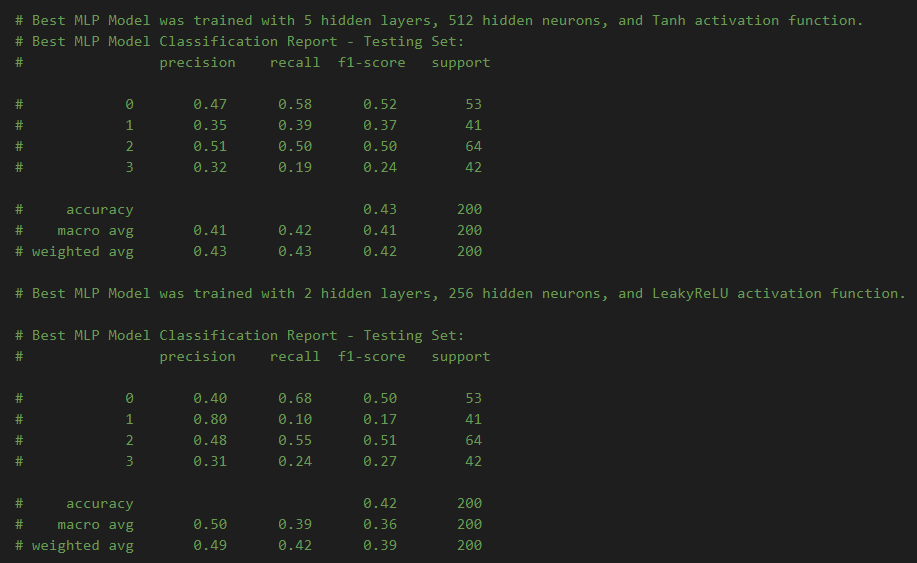


برای نمایش نمودار حالت برگزیده فقط مقادیری که گزارش شده به عنوان ورودی تابع در نظر گرفته شده:





هنگامی که shuffle برابر با True باشد مقادیر مختلفی برای بهترین جواب ارائه میشود:



با شافل برابر True جواب های بهتری هم به دست اومده!

**گزارش 6**

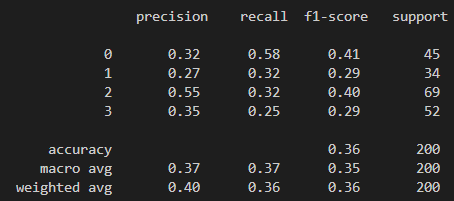
برای قسمت تست بهینه ساز Adam شبکه را با بهترین حالت گزارش شده آموزش میدهیم و در هر بار نرخ بهینه گر Adam را به صورت دستی در کد عوض میکنیم:



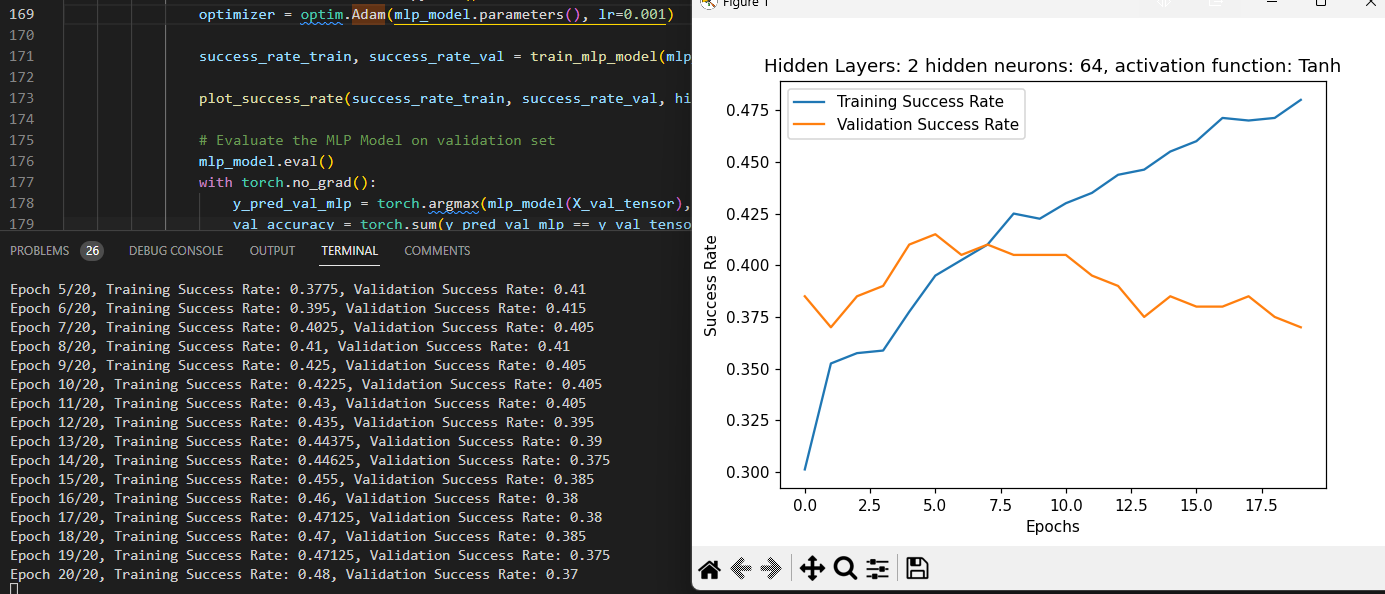
اجرا با 0.01:



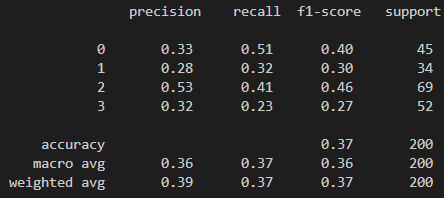
نتیجه گزارش:



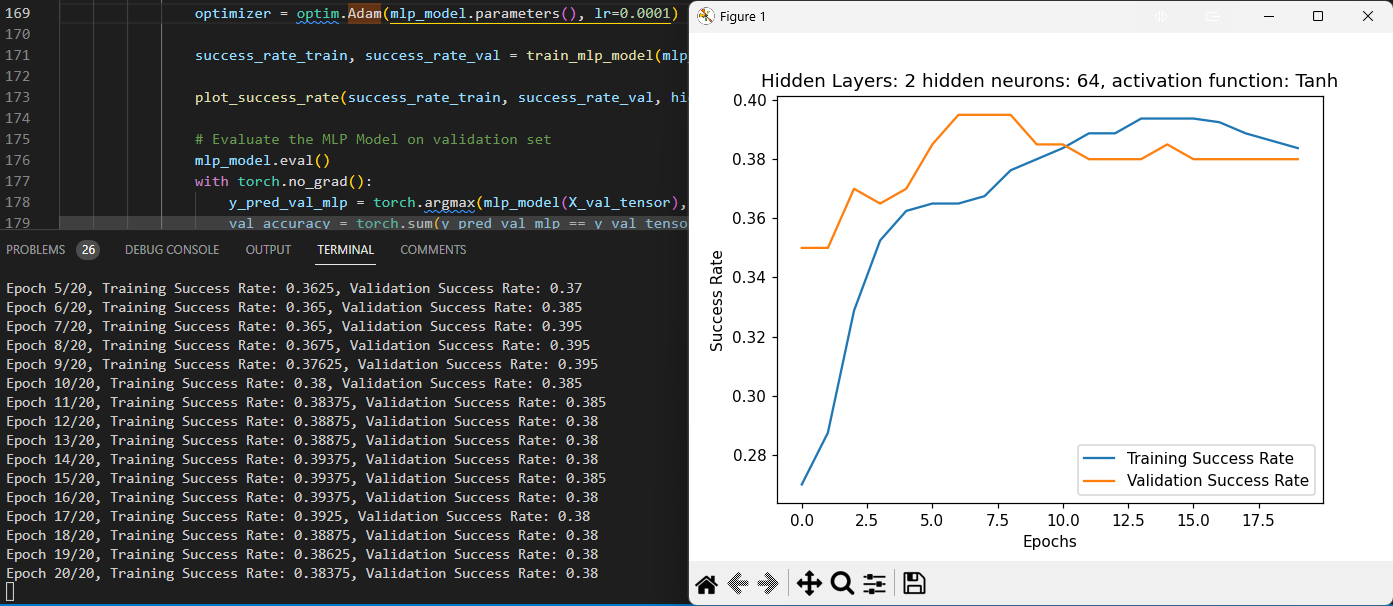
اجرا با 0.001:



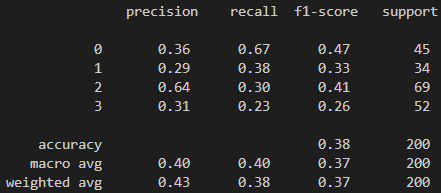
نتیجه:



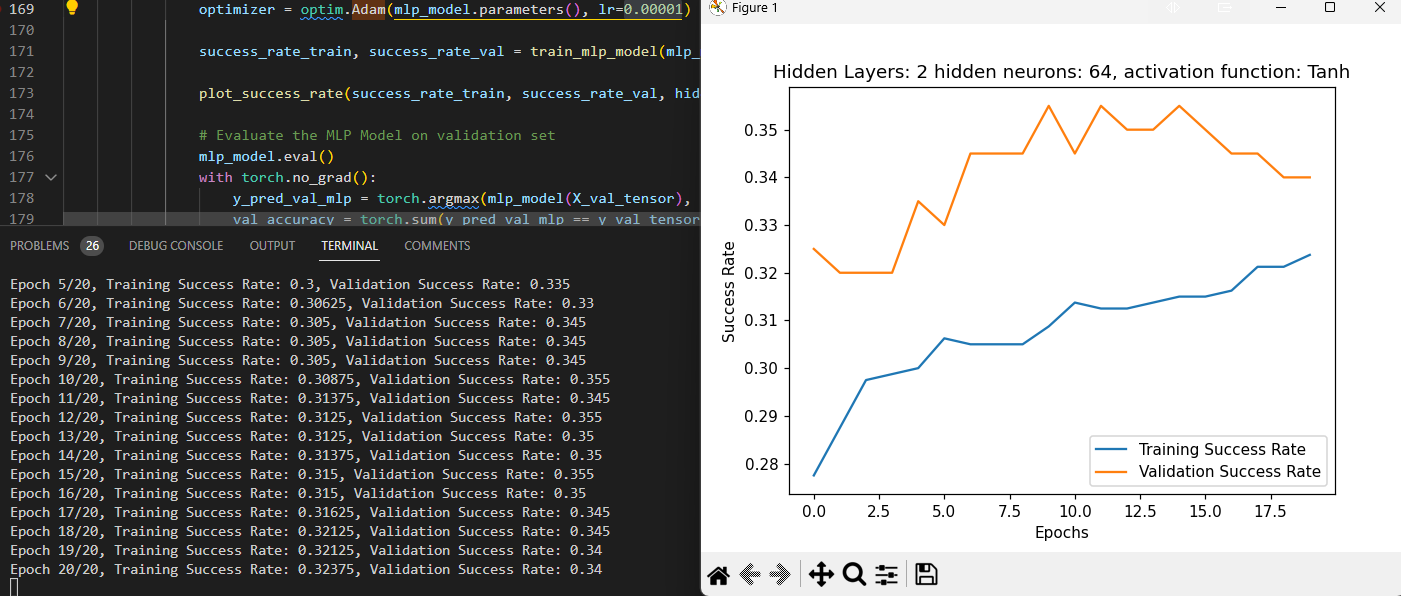
اجرا با 0.0001:



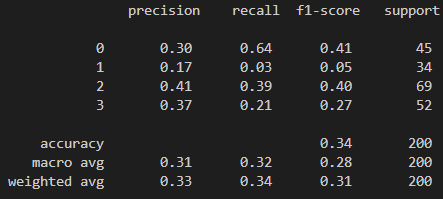
نتیجه:



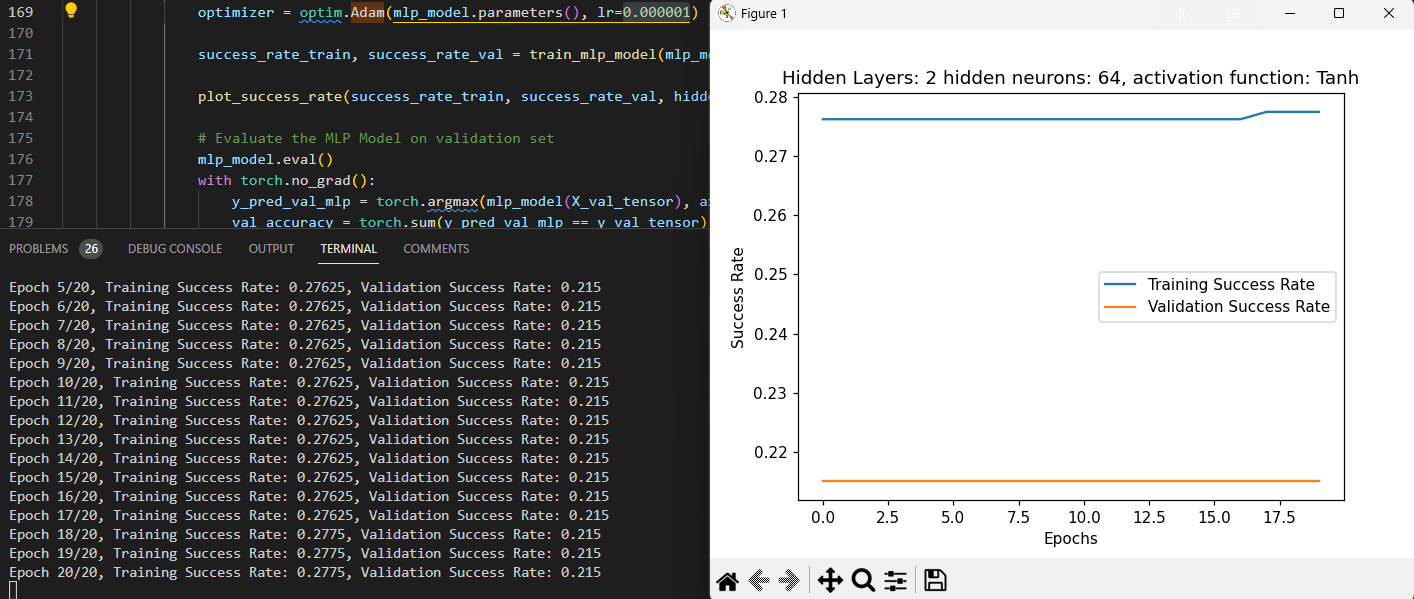
اجرا با 0.00001:



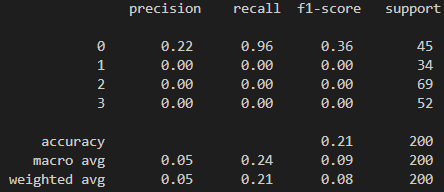
نتیجه:



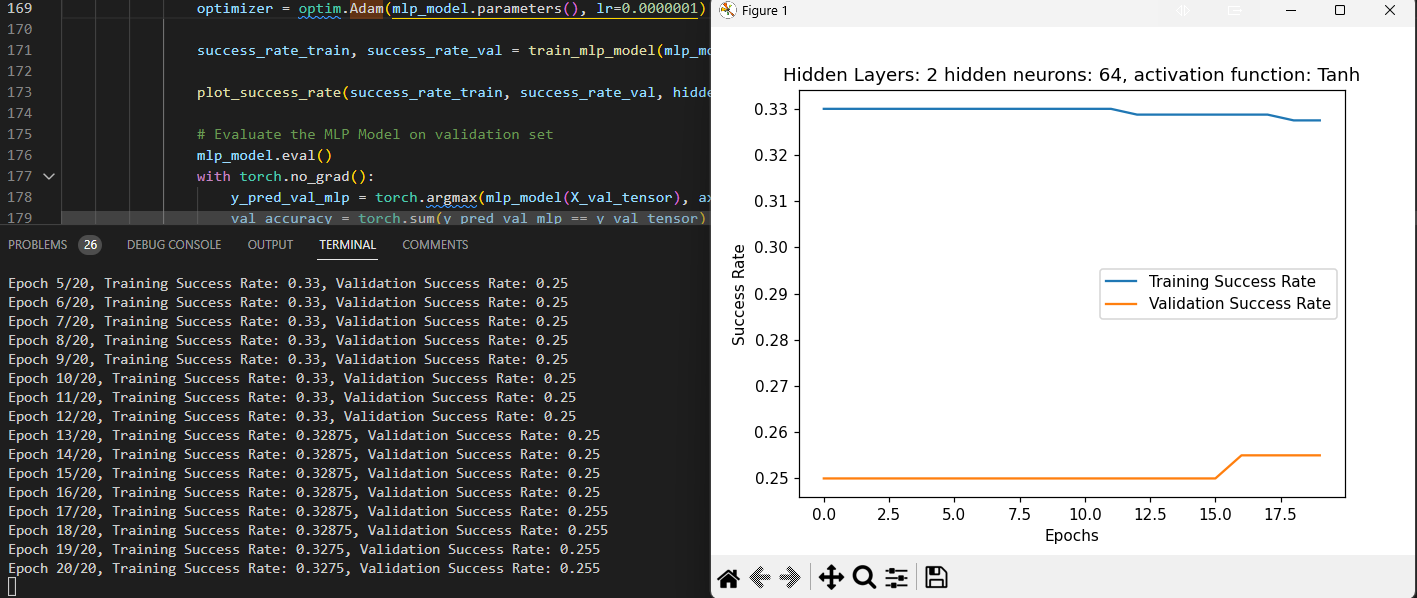
اجرا با 0.000001:



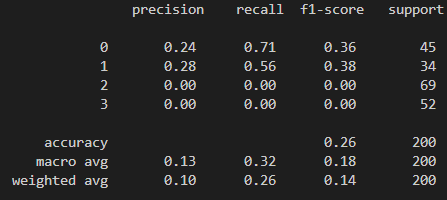
نتیجه:



اجرا با 0.0000001:



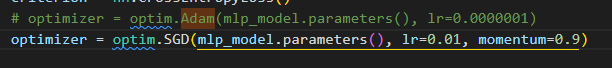
نتیجه:



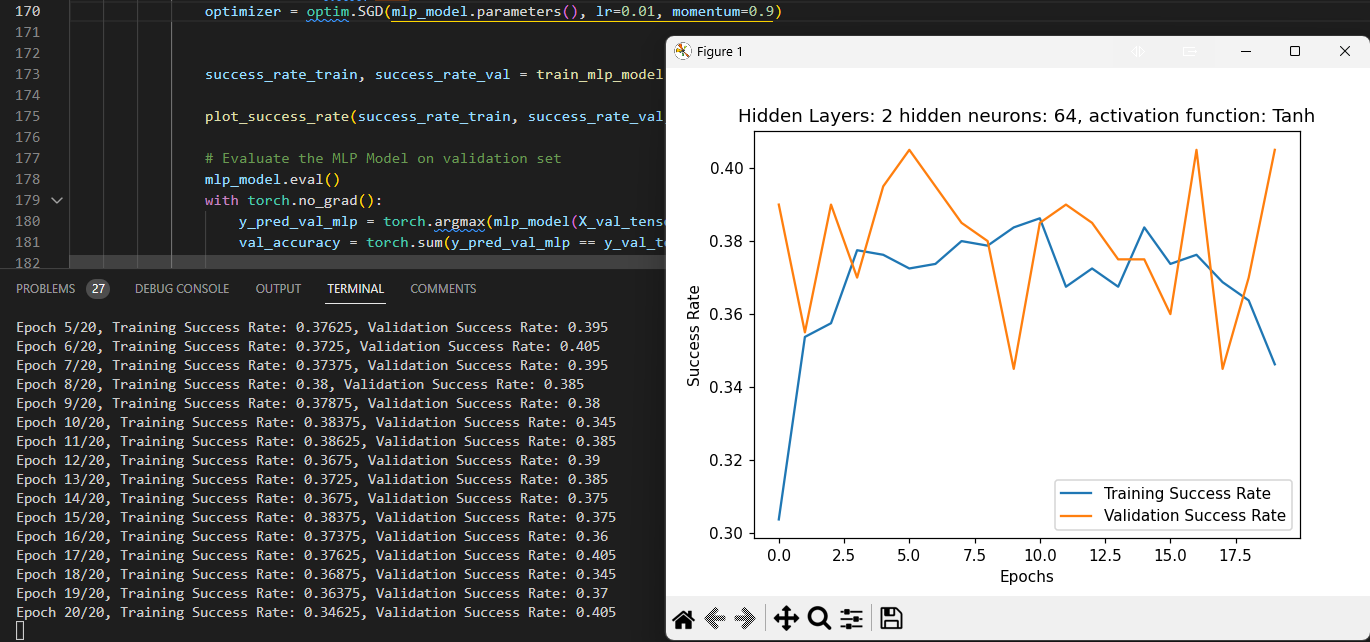
که بهترین دقت برای اجرا با 0.0001 می باشد که برابر 0.38 است.

**گزارش 7**

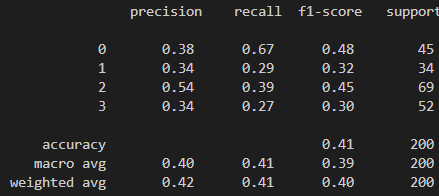
برای آموزش با SGD نیز همین روال انجام شده:



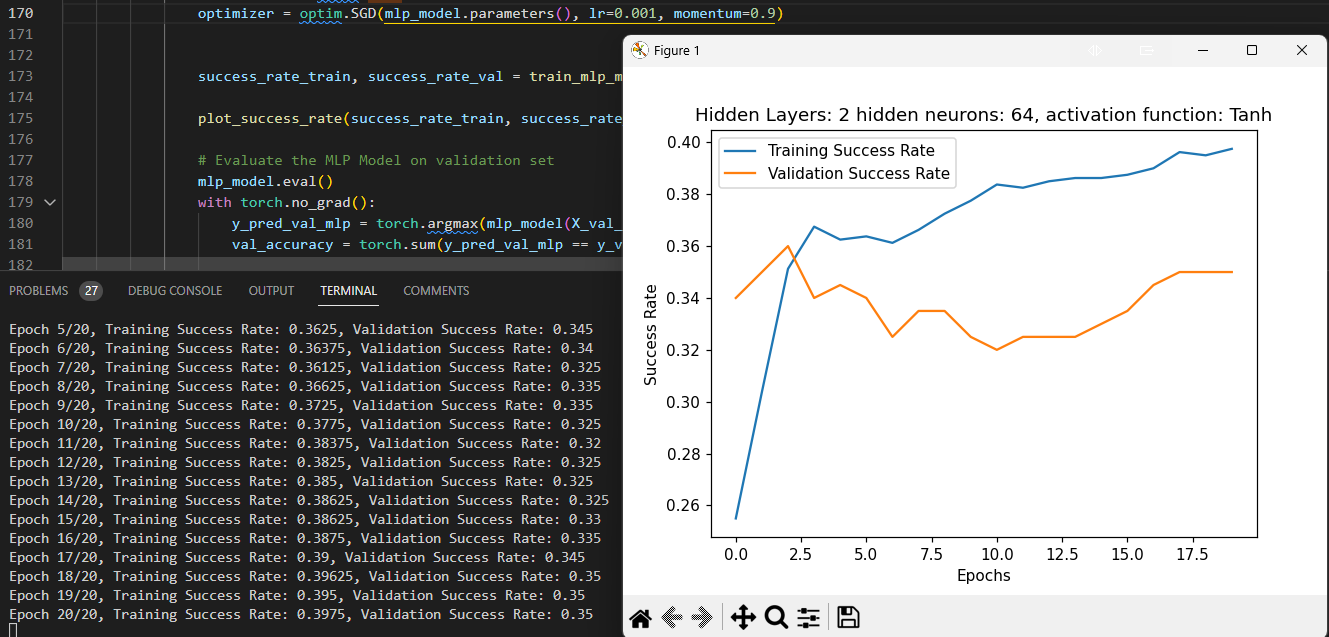
اجرا با 0.01:



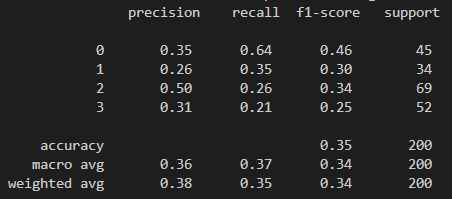
نتیجه:



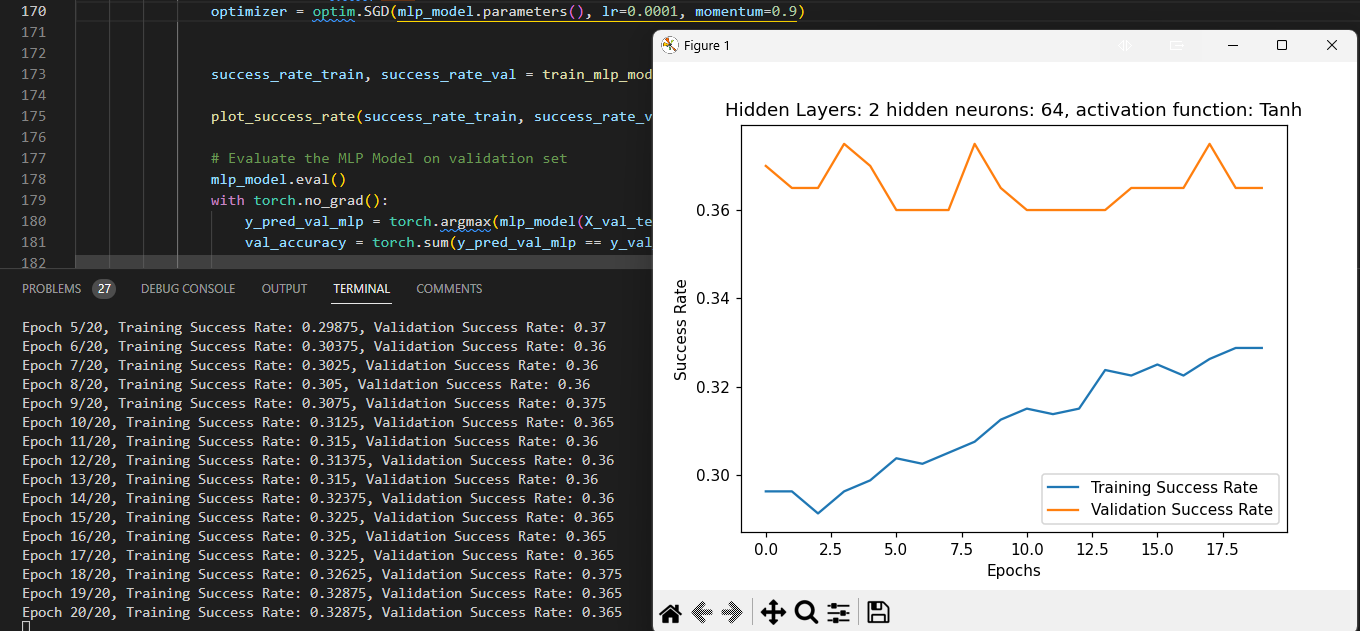
اجرا با 0.001:



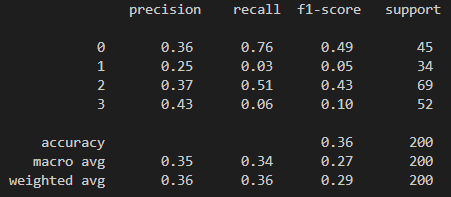
نتیجه:



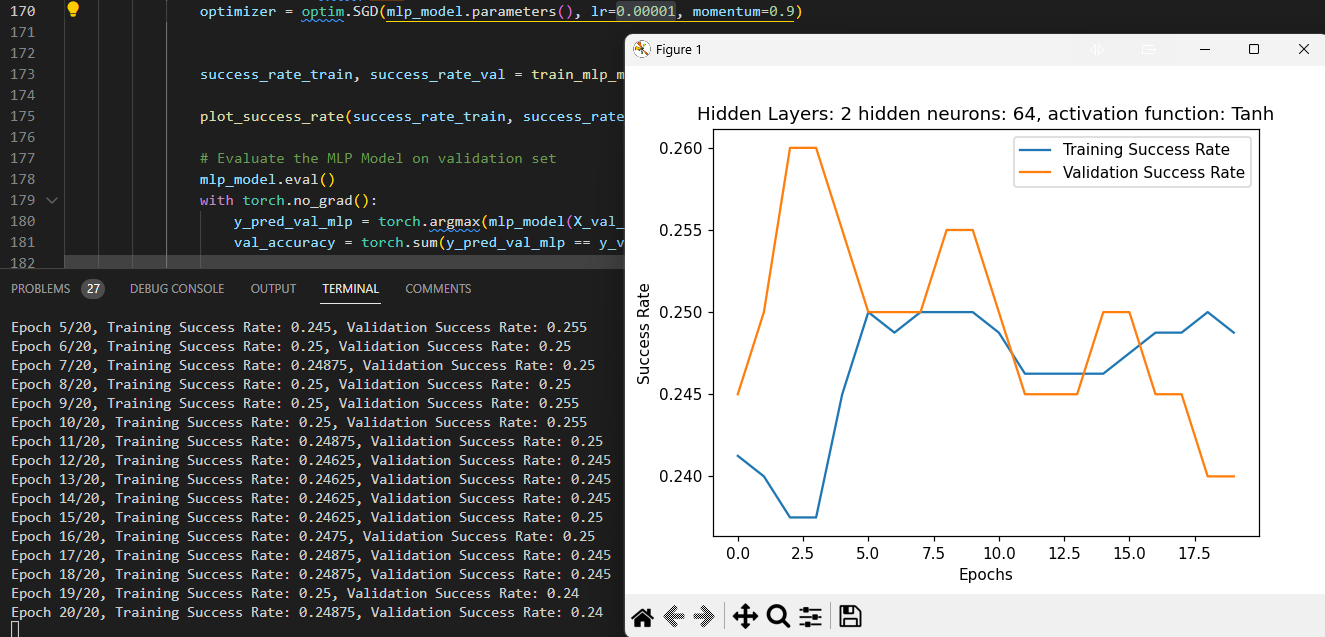
اجرا با 0.0001:



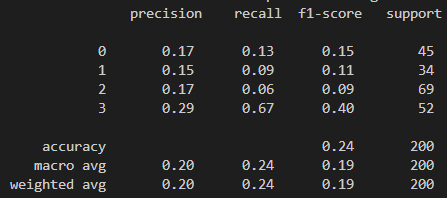
نتیجه:



اجرا با 0.00001:



نتیجه:



همان طور که مشاهده شد بهترین جواب برای اجرا با 0.01 بود و مقدار دقت آن 0.41 بود.

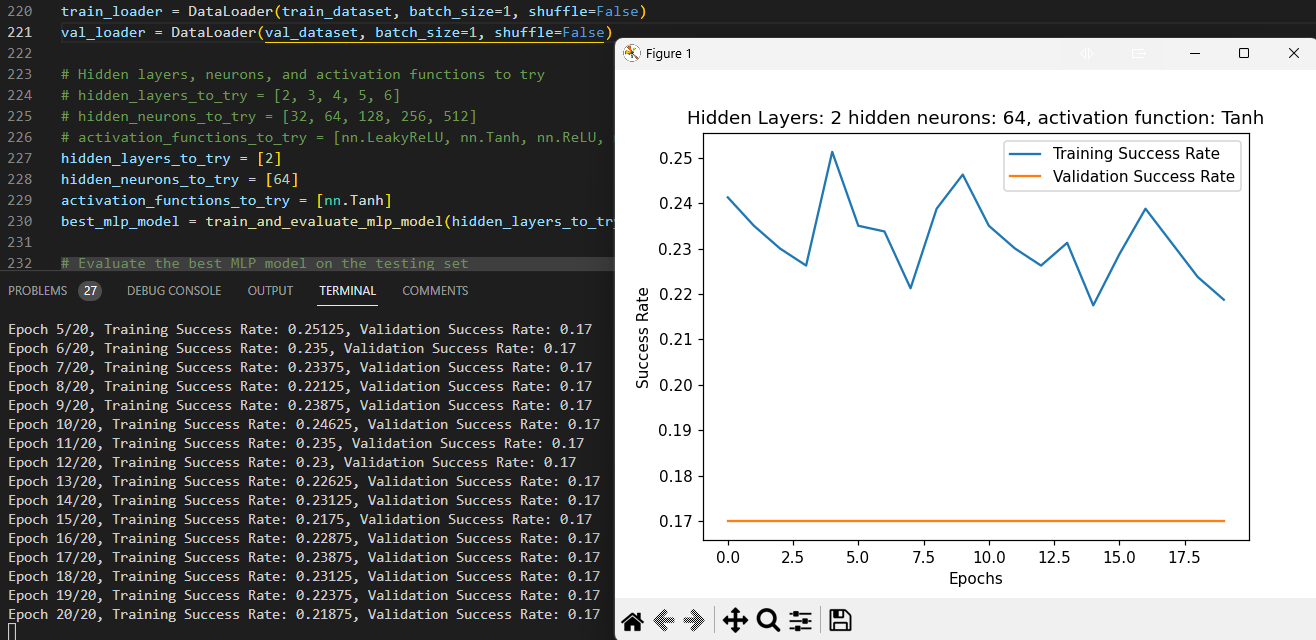
**گزارش 8**

طبق اجرا بهینه گر SGD نسبت به بهینه گر Adam جواب بهتری داد البته ممکن است با ترکیب بهینه گر با تعداد لایه و نورون و فعال ساز جواب بهتری به دست بیاید.

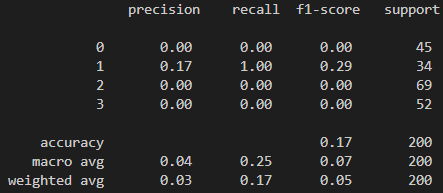
**گزارش 9**

برای تست batch\_size هم به همین شکل عمل میکنیم:

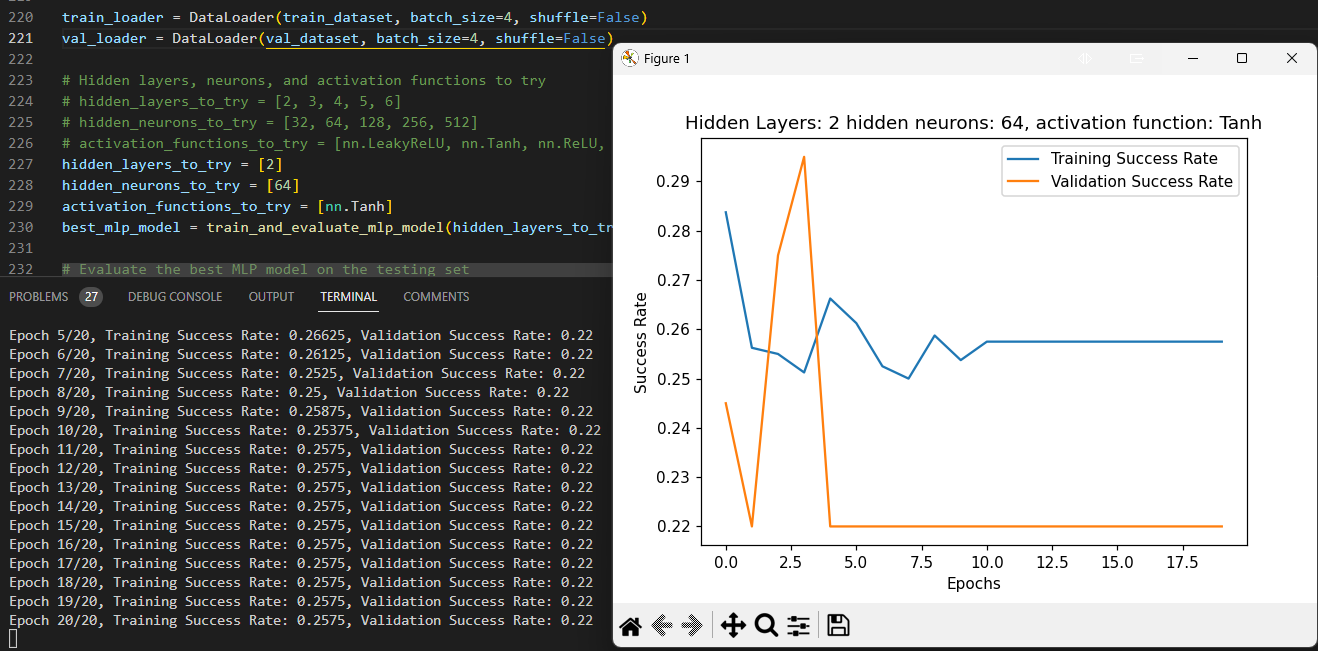
اجرا با 1:



نتیجه:



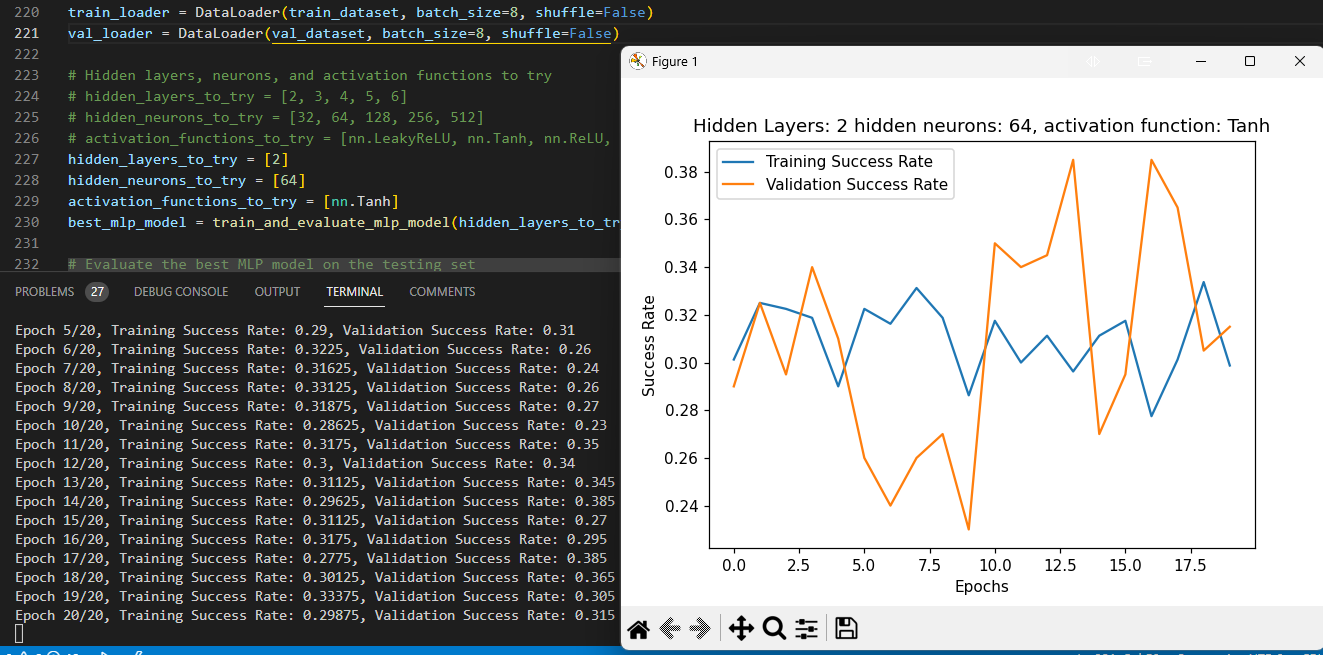
اجرا با 4:



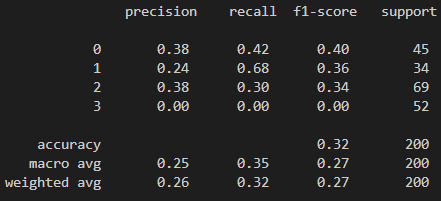
نتیجه:



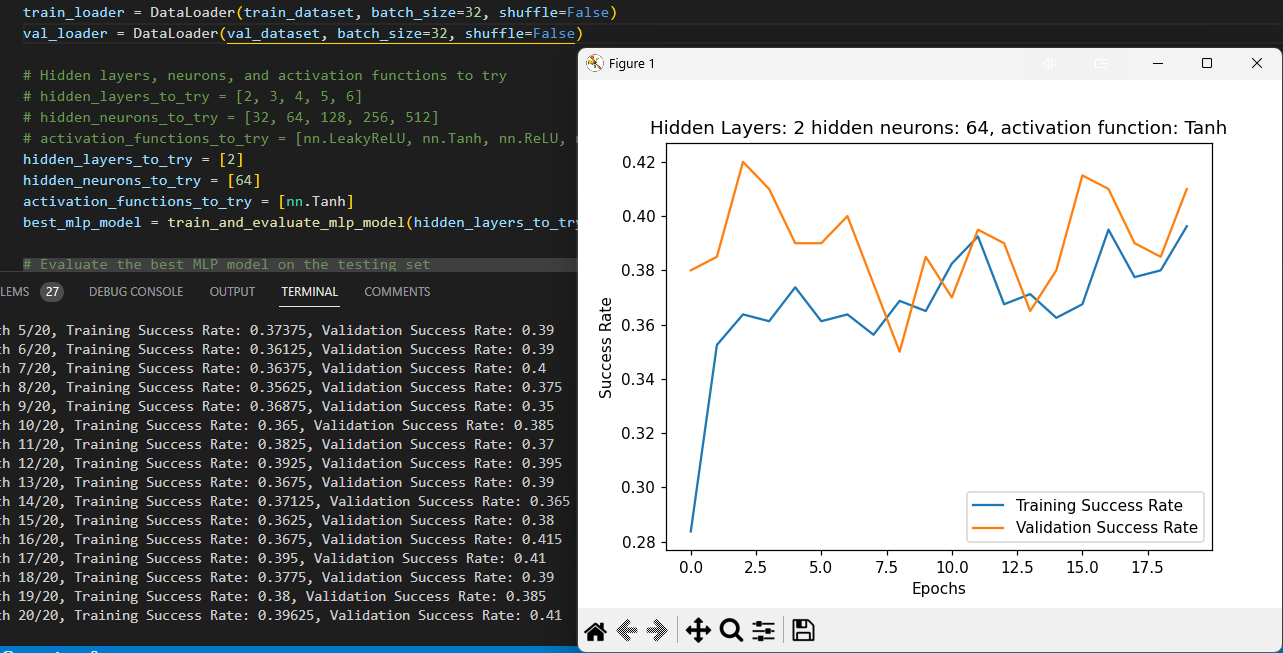
اجرا با 8:



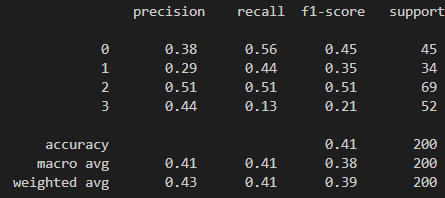
نتیجه:



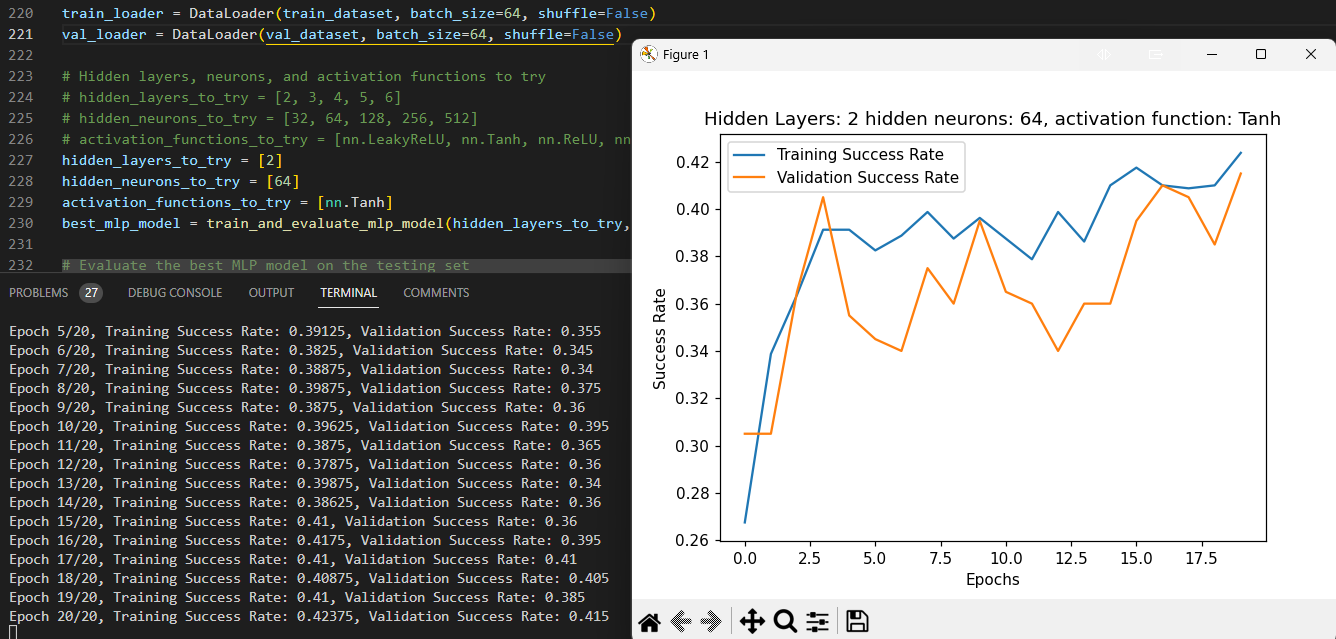
اجرا با 32:



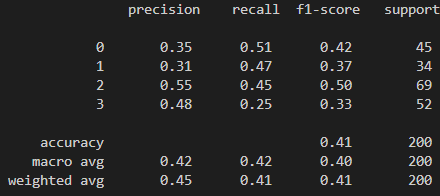
نتیجه:



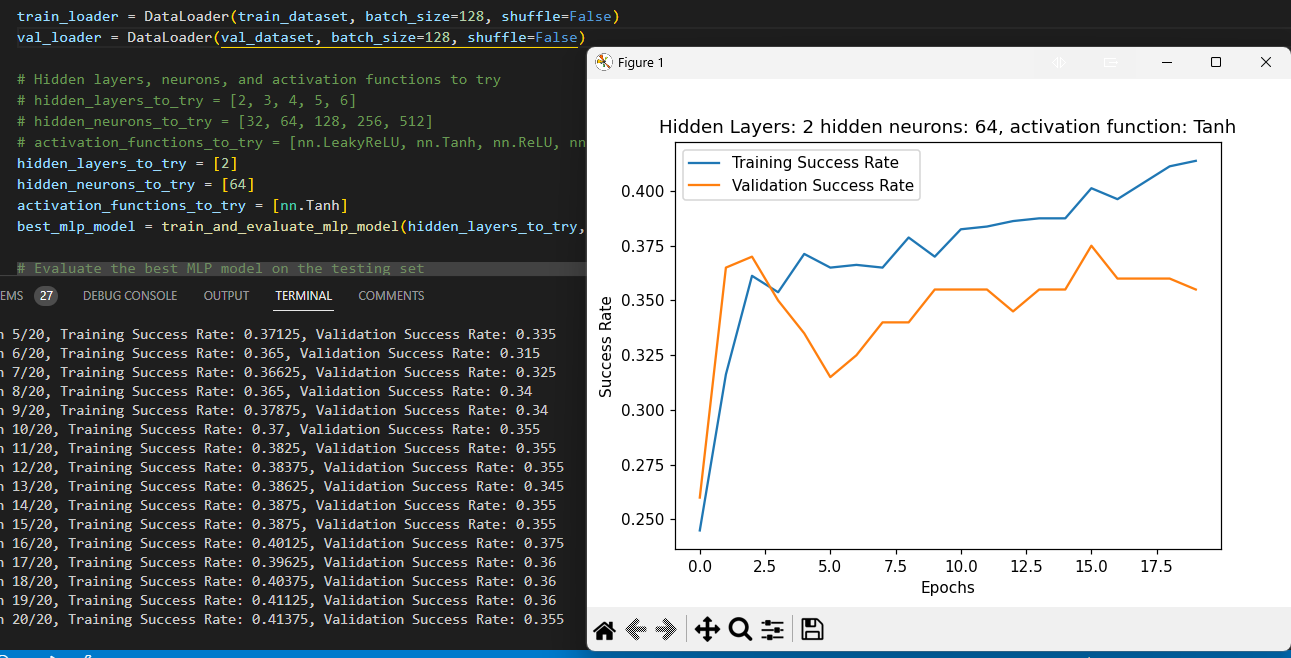
اجرا با 64:



نتیجه:



اجرا با 128:



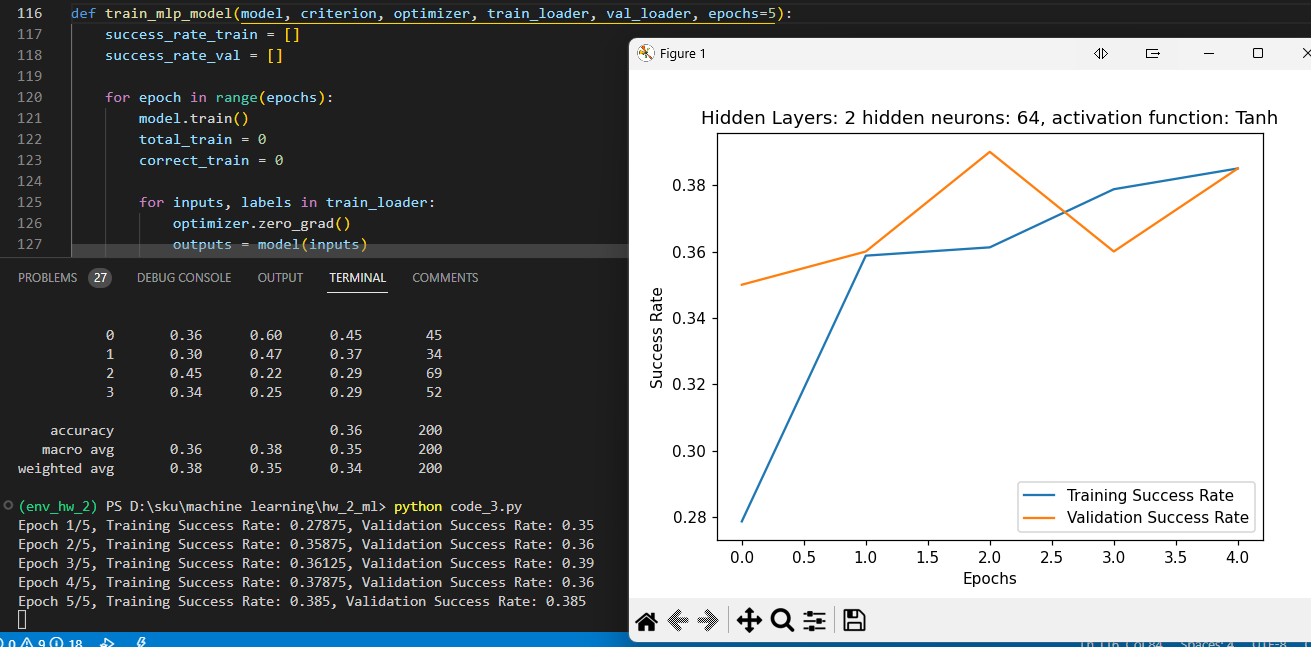
نتیجه:



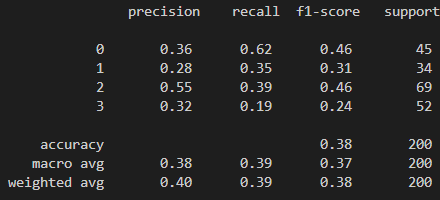
**گزارش 10**

آموزش با تعداد دور های متفاوت:

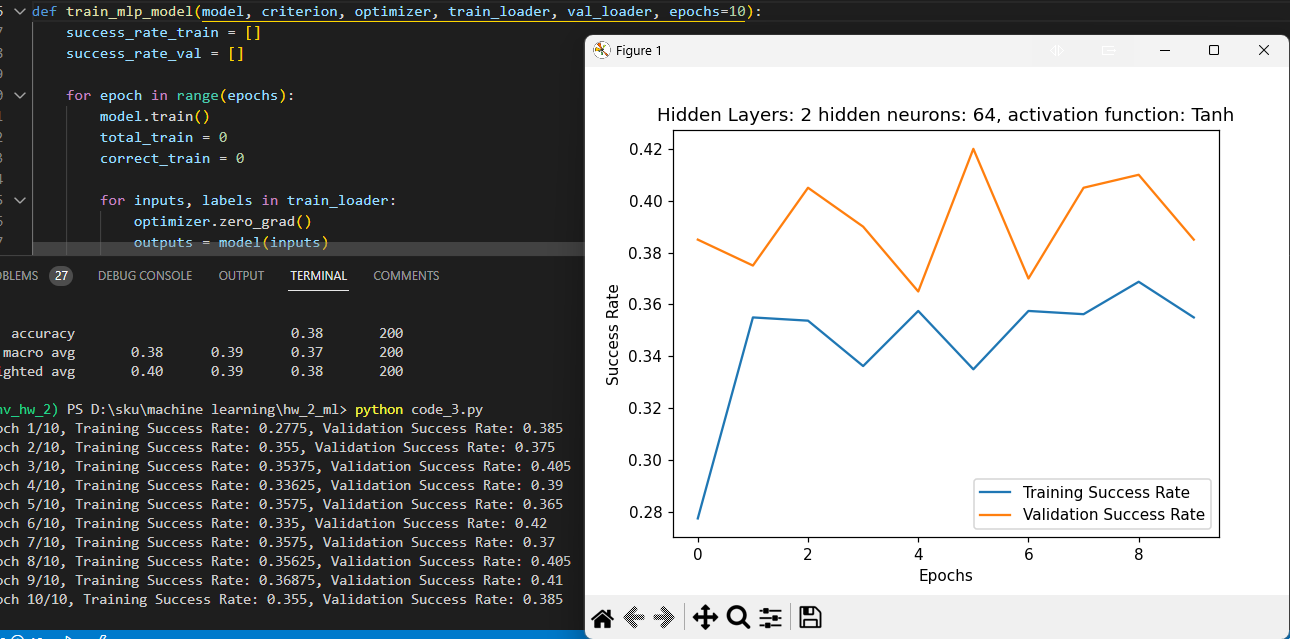
اجرا با 5:



نتیجه:



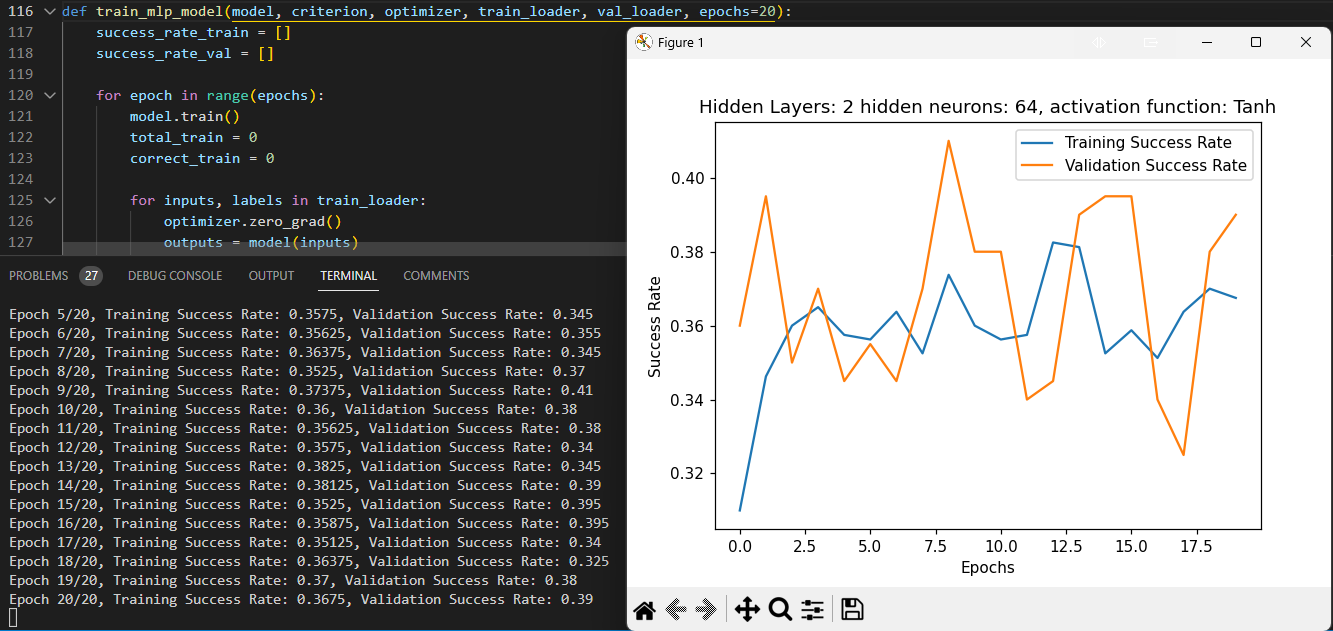
اجرا با 10:



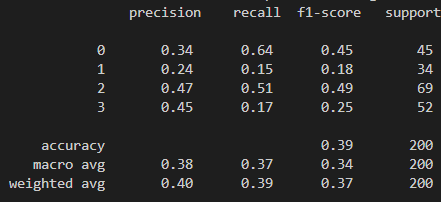
نتیجه:



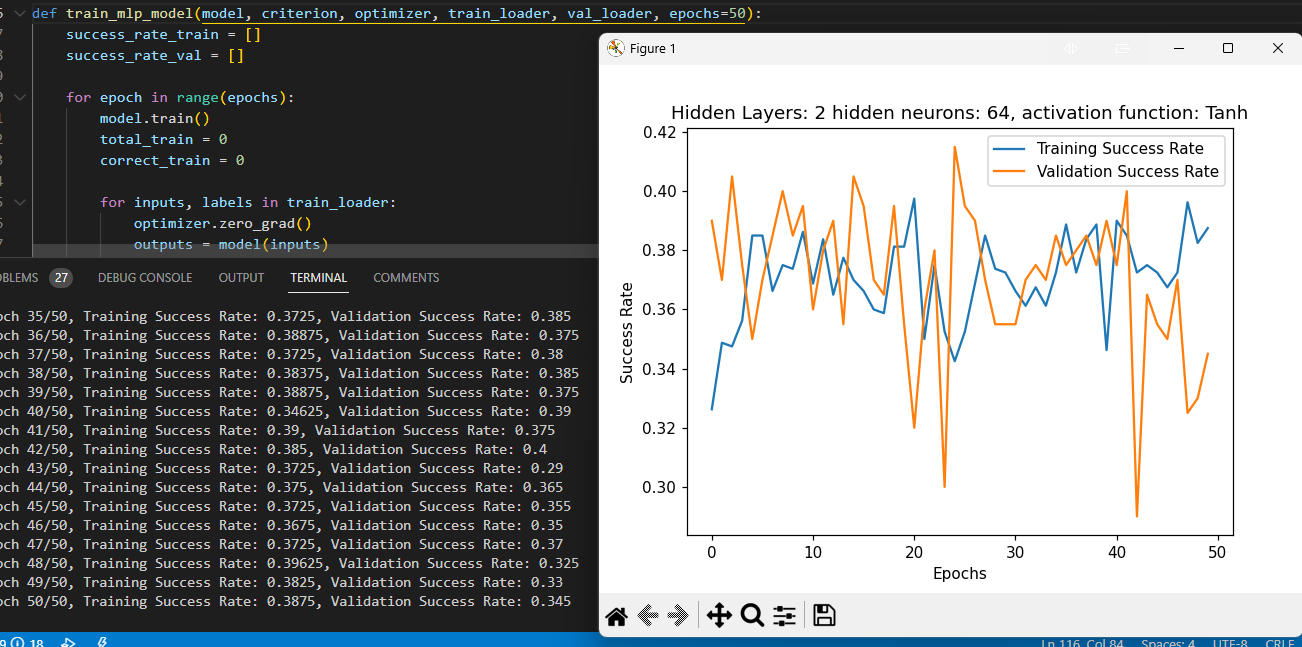
اجرا با 20:



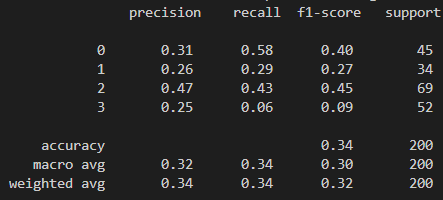
نتیجه:



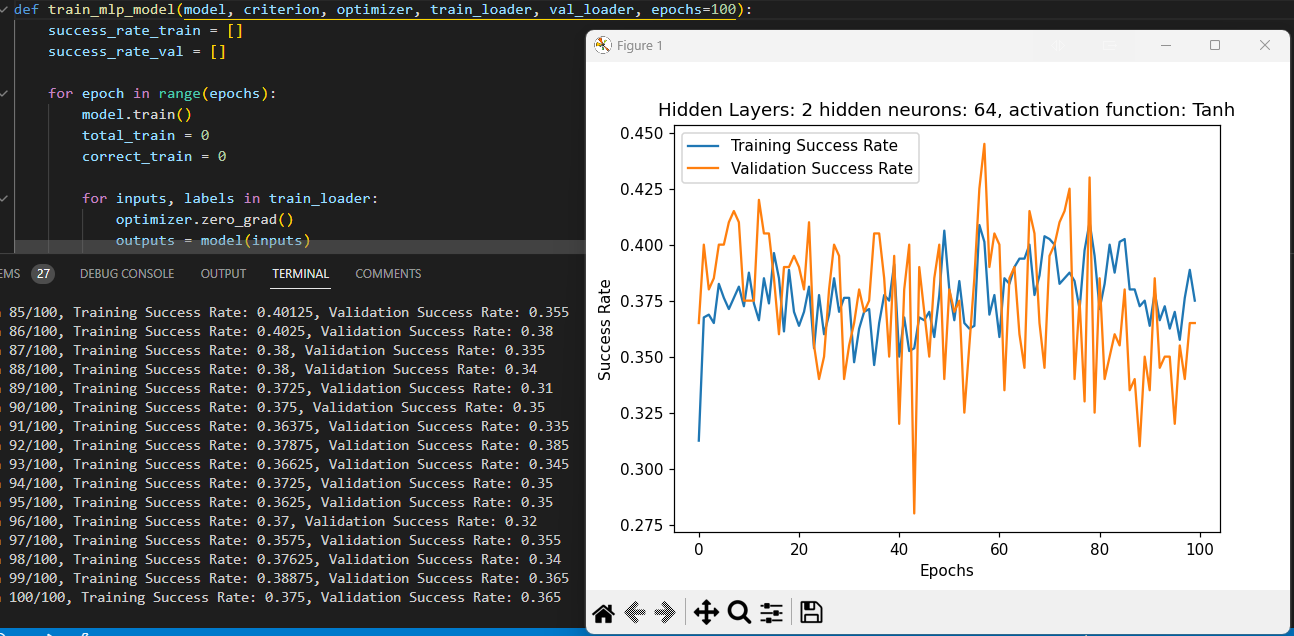
اجرا با 50:



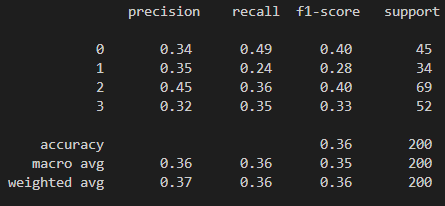
نتیجه:



اجرا با 100:



نتیجه:



**گزارش 11**

رابطه‌ای بین batch\_size و epochs

Batch Size (اندازه دسته): این پارامتر نشان دهنده تعداد نمونه‌های داده که به عنوان ورودی به مدل داده می‌شود در هر مرحله است. بزرگتر کردن batch\_size منجر به افزایش سرعت آموزش می‌شود، زیرا محاسبات موازی‌تر انجام می‌شود. اما افزایش این مقدار ممکن است منجر به حافظه‌گیری زیاد و کاهش تنوع در به‌روزرسانی وزن‌ها شود.

Epochs (تعداد دوره‌ها): یک epoch به مرور کامل دیتاست آموزشی می‌پردازد. افزایش تعداد epochs به معنای افزایش تعداد مراتبی است که مدل داده‌ها را مشاهده و به‌روزرسانی وزن‌ها را انجام می‌دهد. افزایش تعداد epochs می‌تواند منجر به بهبود دقت مدل و یادگیری الگوهای پیچیده‌تر شود.

رابطه بین این دو پارامتر به این صورت است که با افزایش اندازه دسته (batch\_size)، معمولاً تعداد دفعاتی که مدل به تمامی داده‌های آموزشی دسترسی پیدا می‌کند (تعداد epochs) کاهش می‌یابد. این امر معمولاً به معنای این است که مدل با دیدن هر دسته داده بهتر یاد می‌گیرد و نیاز به تعداد کمتری epochs دارد.

با این حال، این قاعده ممکن است در موارد خاصی تغییر کند، و برخی مدل‌ها یا دیتاست‌ها ممکن است به بهینه‌ترین ترکیب از این دو پارامتر نیاز داشته باشند. در عمل، بررسی و تنظیم این پارامترها نیاز به آزمون و خطا و تجربه دارد.

با تغییر مقادیر بهترین جوابی که حاصل شد دقت 0.43 بود:

