توجه: کد پایتون این پروژه از سه کلاس FirstPart.py و SecondPart.py و ThirdPart.py تشکیل شده است که هر سه در کلاس Main، import شده اند و برای شروع عملیات کافی است دستوری که تابع startClassification آن کلاس را صدا میزند uncomment کنید.

بخش اول:

در کلاس FirtsPart تابعی به نام generateSamples وجود دارد که سه پارامتر ورودی میگیرد؛ پارامتر اول یکی از سه رشته blob یا moon یا circle است که به این منظور استفاده میشود که داده ها به چه شکل و در واقع با استفاده از چه تابعی ساخته شوند؛ پارامتر دوم نشانگر نوع تابع هسته استفاده شده است که به یکی از سه حالت linear یا rbf یا poly یا sigmoid است و پارامتر سوم نیز میزان نویز را نشان میدهد که عددی رندوم و به صورت uniform در بازه 0 و 1 است. (البته برای حالت blob چون تابع آماده ورودی نویز ندارد به عنوان cluster_std آن دو برابر همین مقدار نویز داده میشود)

در این تابع ابتدا چک میشود که kernel ورودی معتبر باشد و سپس بسته به پارامتر اول(sampleType) داده های ساخته میشوند؛ در متغیر X مختصات نقاط ساخته شده و در Y نوع آنها(به صورت 0 یا 1) وجود دارد؛ در ادامه با استفاده از این مقادیر دو کلاس مختلف با تابع scatter و با رنگ، label و marker جدا و مشابه داک داده شده به plot اضافه میشوند، خروجی این تابع نیز مدلی است که با استفاده از کلاس SVM و هسته مورد نظر fit شده اند.

خروجی تابع بالا به عنوان ورودی به تابع دیگری به نام plotMargin داده میشود که با استفاده از توابع آماده، خط جداکننده(البته نه الزاما خط راست!) را مشخص کند.

نکته: در داک تمرین گفته شده بود که برای این بخش تست کیس نیاز است؛ به همین منظور با ورودی دادن پارامتر های مختلف که در بالا گفته شد میتوان تست کیس های مختلف ساخت و چون نویز نیز به طور رندوم محاسبه میشود حالت های ساخته شده متفاوت خواهند بود.

نکته2: پس از آزمایش شکل های مختلف با هسته ها و نویز های مختلف در حالت کلی و به صورت تجربی به نتایج زیر رسیدم:

1- برای حالت blob بهترین هسته rbf است و سپس poly و linear و در رتبه آخر sigmoid؛ البته گاهی اوقات poly فضای خالی بین نقاط کلاس ها را به دو قسمت مساوی تقسیم نمیکند.

2- برای حالت moon بهترین هسته با اختلاف rbf است؛ poly نسبت به linear بهتر است ولی در مجموع هیچ یک از این دو عملکرد خوبی ندارند و در رتبه آخر نیز sigmoid قرار دارد که کلا درست جدا نمیکرد.

3- برای حالت circle نیز مانند حالت moon است با این تفاوت که rbf نیز گاها دچار مشکل شدید میشود و شاید بتوان گفت sigmoid در رتبه دوم قرار میگیرد.

در کل پارامترهای دیفالت استفاده شدند.

بخش دوم:

در این بخش از پایگاه داده USPS استفاده شده است که تصاویر آن در 10 کلاس مختلف که هر کدام مربوط به یک عدد هستند تقسیم بندی میشوند و با استفاده از داده های فولدر train این یادگیری انجام میگیرد و در نهایت با استفاده از داده های test عملکرد آن بررسی میشود.

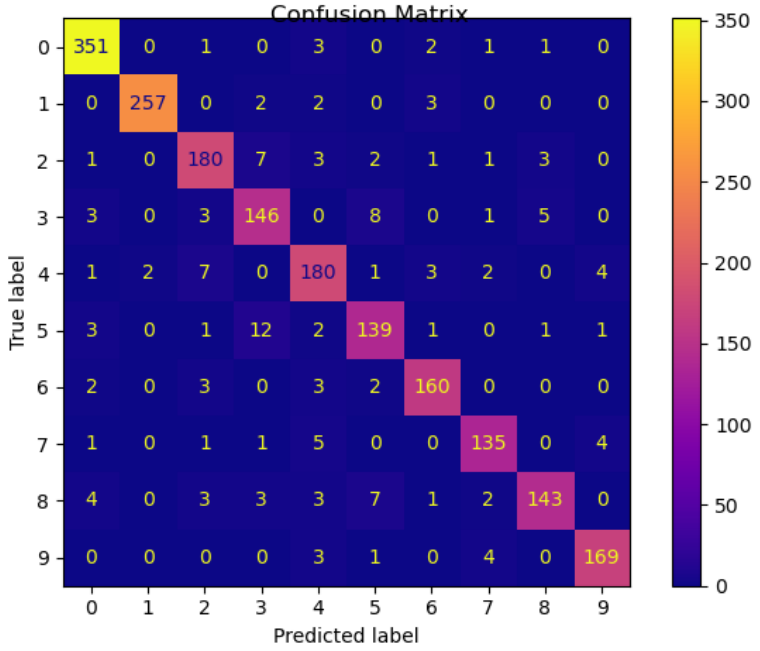
کلیت عملکرد این بخش که در کد تحت فایل SecondPart.py قرار دارد به این شکل است که ابتدا تابعی صدا زده میشود تا اطلاعات تصاویر مربوط به پایگاه داده USPS که در فولدر images قرار دارند خوانده شده و تحت عنوان 4 لیست x_train,y_train,x_test,y_test قرار گیرند؛ محتوات داخل y ها در واقع همان عددی است که آن تصویر نشان میدهد و از اسم این تصاویر استخراج شده است و به عنوان کلاس هر تصویر استفاده میشود.

متغیر img آرایه ای دوبعدی است که به صورت grayScale تصاویر را میخواند و سپس عملیاتی روی آن انجام میشود تا به شکلی تبدیل شود که در ادامه برای fit کردن داده ها با استفاده از Svm مشکلی پیش نیاید.

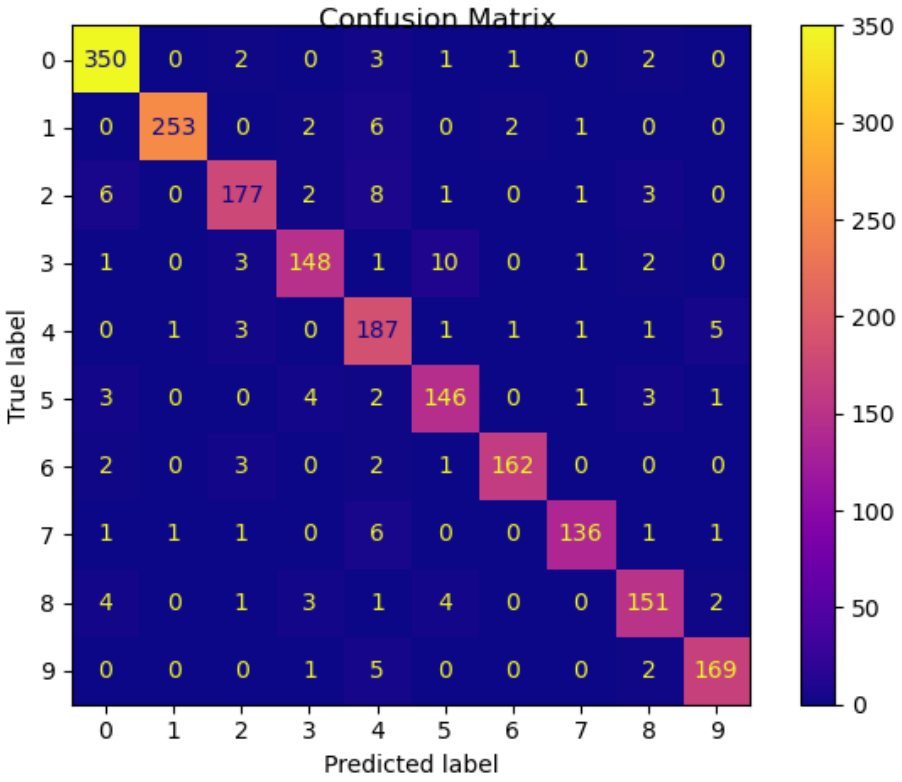
سپس تابع predit صدا زده میشود که عملیات پیشبینی کلاس هر یک از تصاویر فولدر test (که در x_test قرار دارند) را پیشبینی کند؛ در نهایت کلاس هر یک از این مقادیر با کلاس واقعی هر یک که در y_test قرار دارند مقایسه شده نتیجه نهایی هم به صورت متنی و هم به صورت نمودار نمایش داده میشود. (مقدار دقت نیز هم به ازای هر عدد و همچنین برای کل عملیات نیز در کنسول چاپ میشوند)

در ادامه عکسی از نتایج به دست آمده با استفاده از هر یک از هسته تا قرار دارد:

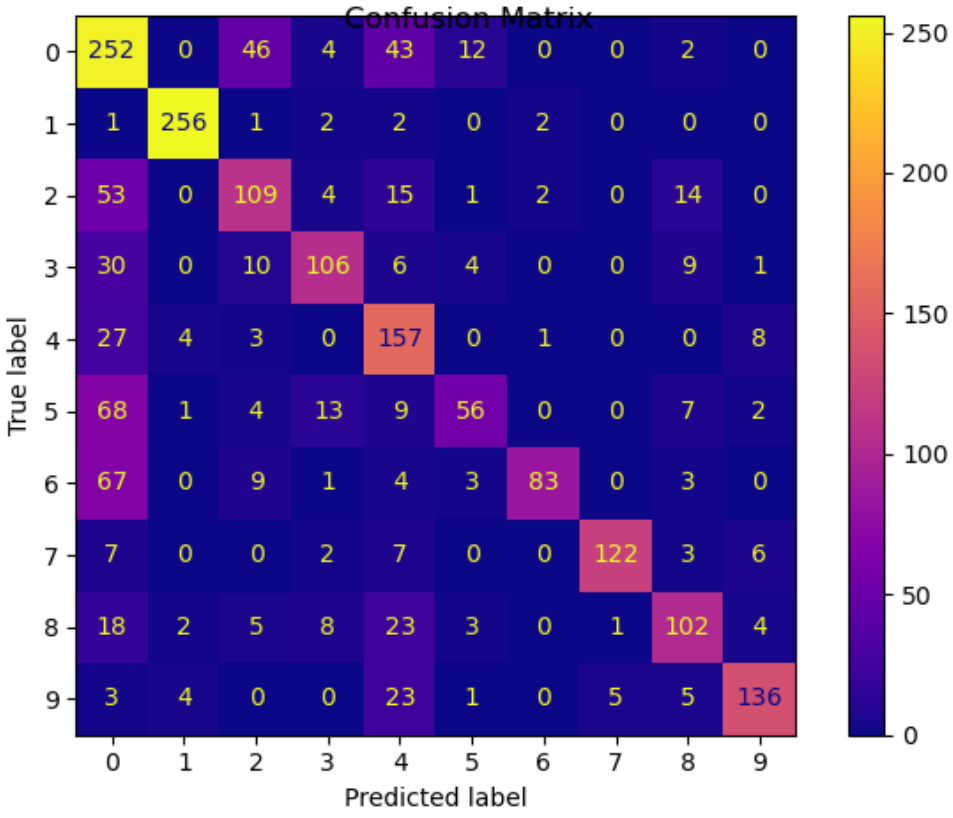
linear -1



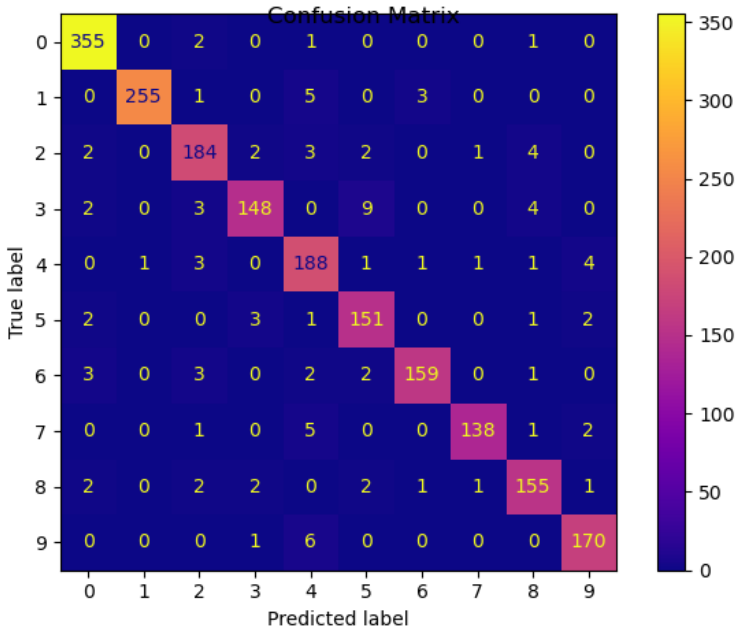
poly-2



sigmoid-3

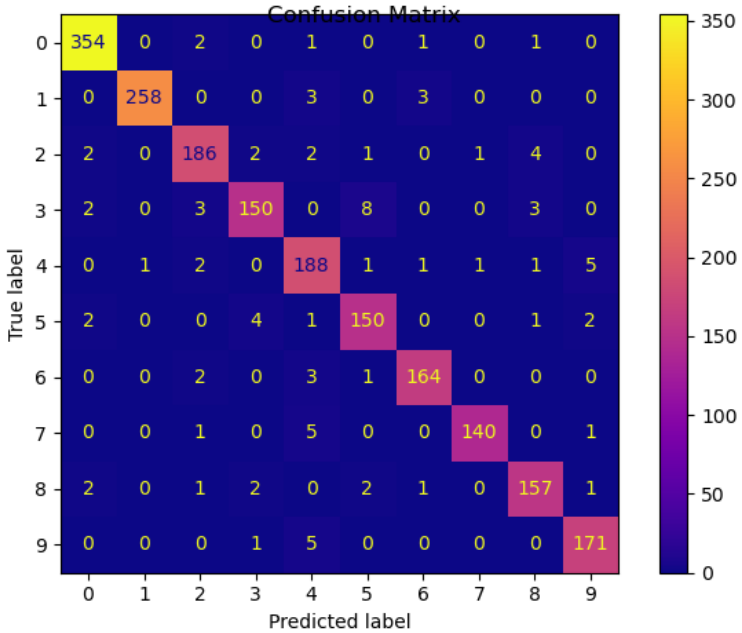


rbf-4



نتیجه: به طور قطع میتوان گفت که هسته sigmoid بدترین خروجی را دارد؛ در رتبه اول میتوان rbf را قرار داد اما poly و linear رقابت نزدیکی برای رتبه دوم دارند و میتوان رتبه دوم مشترک را به آنان اختصاص داد. در ادامه تأثی تغییر پارامترها برای هسته rbf بررسی شده است.

پارامتر های probability و random_state را تغییر دادم اما هیچ تغییری حاصل نشد و نتیجه عینا همان قبلی بود. همچنین پارامتر gamma را از حالت دیفالت تغییر داده و برابر عددی float قرار دادم (0.0001) که برای تقریبا تمامی تصاویر تست، 0 را پیشبینی کرد؛ اگر این پارامتر برابر auto باشد برای دقیقا تمامی تصاویر 0 را پیشبینی میکند. پارامتر آخری که تغییر دادم پارامتر C بود که به طور دیفالت برابر 1 است و آن را برابر 10 قرار دادم و نتیجه زیر به دست آمد:



که این نتیجه نیز تقریبا همان نتیجه با C=1 است و تغییر چندانی نداشتیم

نتیجه نهایی: تغییر این پارامتر ها نشان داد بهترین حالت همان حالت دیفالت است و بهتر است فقط هسته را مشخص کرده برای بقیه پارامتر ها را به حالت دیفالت خود این تابع اعتماد کنیم! همچنین دقت برای داده های تست در حدود 95 درصد بود. همچنین برای داده های train دقتی در حدود 99 درصد داشتیم که دچار بیش برازش شده است البته چون دقت داده های تست مقدار قابل قبولی است فکر نمیکنم این اتفاق مشکل آنچنان بزرگی باشد!

نتیجه مقایسه با شبکه عصبی:

در این بخش عملکرد svm با استفاده از rbf با عملکرد شبکه عصبی در حالتی که solver=lbfgs باشد مقایسه شده است:

در حالتی که برای شبکه عصبی دو لایه پنهان و 100 نورون در هر لایه داشته باشیم، عملکرد svm بهتر است؛ در حالتی که 1000 نورون در هر یک از این دو لایه قرار داشته باشد عملکرد تقریبا یکسانی دارند و طبیعتا به ازای لایه ها و نورون های بیشتر، عملکرد شبکه عصبی بهتر خواهد بود.

البته زمان اجرای دسته بندی برای Svm سریعتر از شبکه عصبی است (زمانی که تعداد لایه ها و نورون ها بیشتر از مقادیر بررسی شده باشد) و این را میتوان نقطه قوت svm در مقابل شبکه عصبی دانست!

بخش سوم:

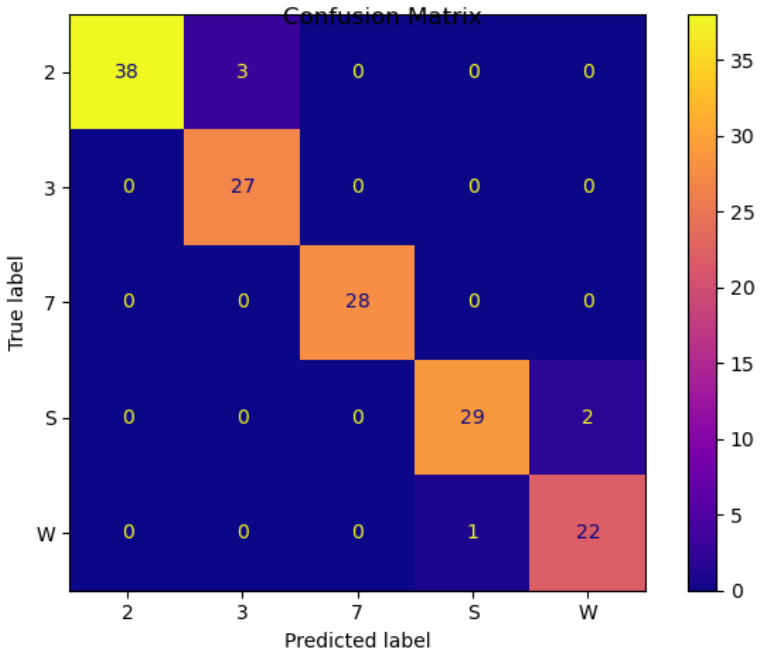
در این بخش و در کلاس ThirdPart.py به این شکل عمل شده است که ابتدا تمام تصاویر خوانده شده و کلاس آنها که همان نام فولدر آنهاست مشخص شده و این مقادیر در دو آرایه به نام های input_images و input_images_label قرار گرفته اند؛ سپس آن دو آرایه به دو بخش داده های آموزشی و آزمایشی تقسیم شده اند و سپس عملیات یادگیری انجام شده است و در نهایت نتایج حاصله به مانند بخش دوم نمایش داده شده اند.

در ادامه به ازای هسته rbf (و بقیه پارامتر ها به صورت دیفالت) به ازای درصد های مختلف تقسیم داده ها به بخش test و train، نتایج حاصله و دقت svm بررسی شده است:

توجه: با توجه به اینکه تابع train_test_split به طور رندوم داده ها را تقسیم بندی میکند پس با هر بار اجرا کردن برنامه نتیجه اندکی تفاوت خواهد کرد

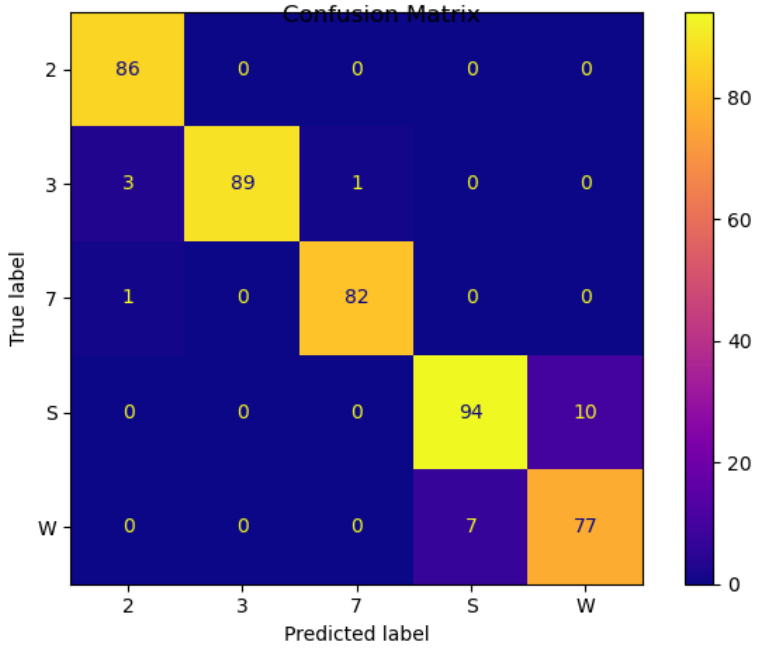
test_size=0.1 -1

در این حالت دقت برای داده های آزمایشی(که عکس آن در ادامه آمده است) برابر با 96 درصد بود. (دقت داده های آموزشی در حدود 97 درصد)



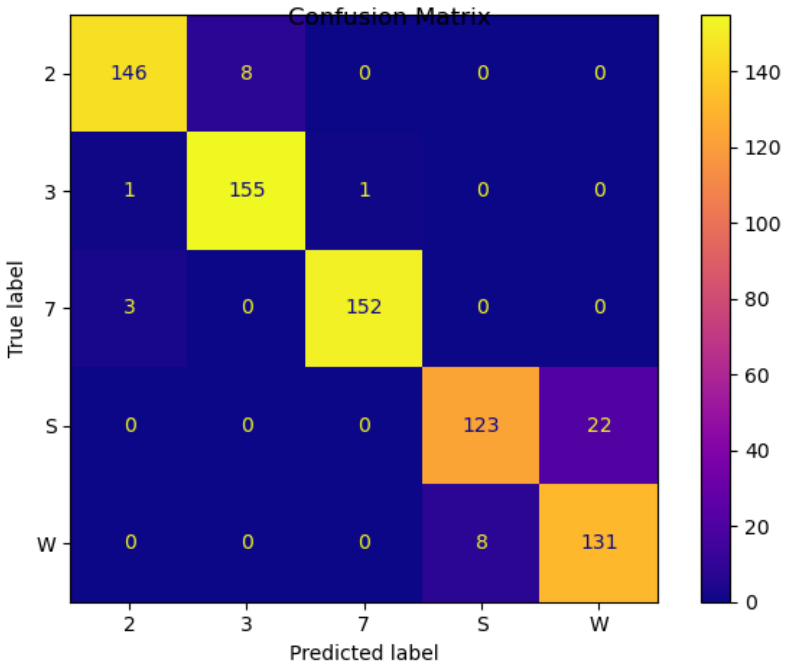
test_size=0.3 -2

در این حالت، دقت برای داده های تست در حدود 95 درصد بود که مشکل اصلی تشخیص دو حرف ص و س از یکدیگر بوده است و تشخیص اعداد مشکل چندانی نداشته است. (دقت داده های آموزشی در حدود 96 درصد)



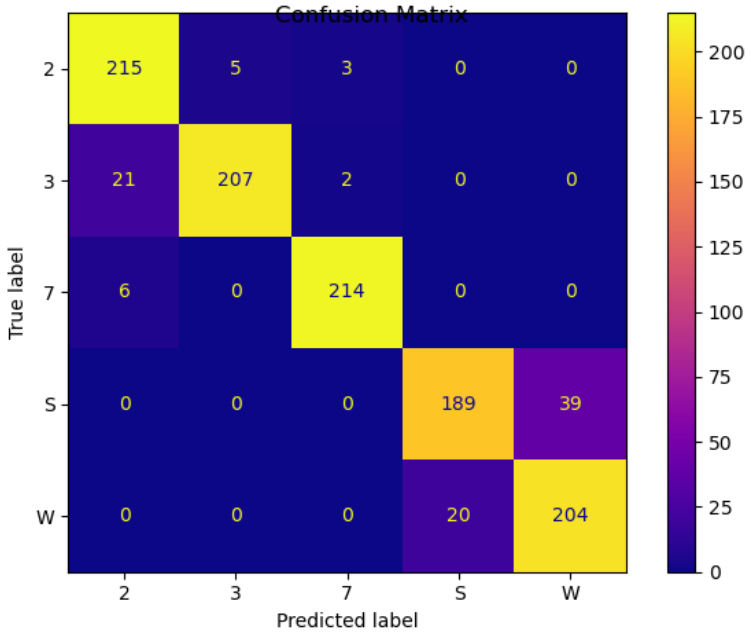
test_size=0.5 -3

در این حالت، دقت برای داده های تست در حدود 94 درصد بود که باز هم مشکل اصلی تشخیص دو حرف ص و س از یکدیگر بوده است و تشخیص اعداد مشکل چندانی نداشته است. (دقت داده های آموزشی در حدود 95 درصد)



test_size=0.75 -4

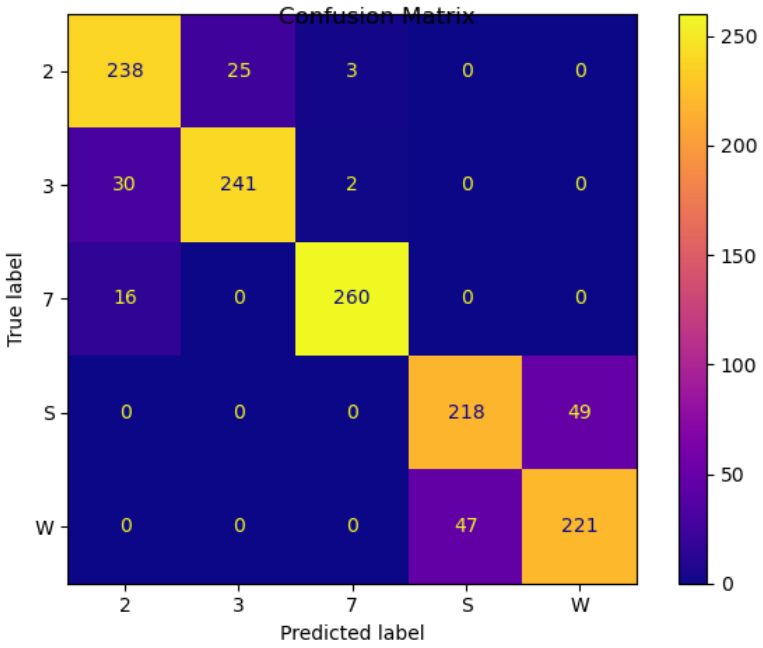
در این حالت، دقت برای داده های تست در حدود 91 درصد بود که اینبار علاوه بر مشکلات س و ص که افزایش یافته است، مشکل تشخیص 2 و 3 نیز تا حدودی مشکل ساز شده است. (دقت داده های آموزشی در حدود 95 درصد)



test_size=0.9 -5

در این حالت، دقت برای داده های تست در حدود 87 درصد بود که اینبار علاوه بر مشکلات س و ص که به شدت افزایش یافته است (و حتی میتوان گفت غیر قابل اطمینان است)، مشکل تشخیص

2 و 3 نیز به مشکل جدی تبدیل شده و علاوه بر تشخیص 2 و 7 نیز با دشواری هایی مواجه شده است. (دقت داده های آموزشی در حدود 97 درصد)



نتیجه نهایی: همانطور که در آزمایش حالت های مختلف مشخص شد، با کاهش تعداد داده های آموزشی درصد درستی نتایج حاصله نیز کاهش پیدا کرد و این به خوبی اهمیت train را مشخص میکند. در حالت کلی و به صورت عملی اگر کدی که برای تشخیص این پلاک ها از روی عکس دوربین ها به کار میرود، با داده های آموزشی به تعداد زیاد train شده باشد با دقت بسیار خوبی میتواند پلاک را از روی عکس تشخیص دهد. همچنین دقت داده های آموزشی در حدود 96 درصد بیه طور میانگین برای تمام حالت ها بود که فکر نمیکنم دچار بیش برازش شده باشد و از این نظر مشکلی وجود ندارد.

نتیجه نهایی 2: شاید اگر بتوانیم کدی داشته باشیم که علاوه بر پلاک، خودرو را هم تشخیص داده و به ازای تمام شماره پلاک های احتمالی (مثلا ص به جای س در صورتی که نویز زیاد باشد و از تشخیص خود مطمئن نباشد) خودرو را نیز با خودرو واقعی همان پلاک ها که در یک دیتابیسی قرار دارد (احتمالا چنین دیتابیسی باید موجود باشد!) مقایسه کند(از نظر نوع و رنگ و ... نباید خیلی انجام این کار سخت باشد!)، با دقت بسیار بیشتری بتوانیم تشخیص دهیم و اشتباهی فرد دیگری را جریمه نکنیم!