.1

الف) در LBP ما میخواهیم اطلاعات ویژگی های مهم محلی را برای هر پیکسل استخراج کنیم. ولی در استفاده از padding میخواهیم اطلاعات مرزی را حفظ کنیم. بنابراین نیازی به استفاده از padding نداریم. چون با اضافه کردن padding تصویر تغییر کرده و اطلاعات محلی اشتباهی بدست میاوریم.

ب) در این الگوریتم خانه های اطراف ماتریس 3 در 3 را بررسی میکنیم اگر بزرگتر یا مساوی مقدار وسط آن باشد، مقدار1 و در غیر این صورت مقدار 0 میدهیم.

3	5	6	3	5	3
5	8	8	5	8	5
5	8	8	5	8	5
5	8	8	5	8	5
3	5	6	3	5	3

.2

در این سوال ابتدا دو descriptor تعریف میکنیم.

Compactness : فشردگی یک شکل میتواند از مقایسه با این مورد بدست بیاید. فرمول :

$$Compactness = \frac{4\pi\ Area}{Perimeter^2}$$

برای محاسبه محیط شکل از ()contourArea استفاده می کنیم و ورودی آن کانتور های تصویر است. برای محاسبه ی premeter از ()arclength استفاده میکنیم که ورودی آن کانتور های تصویر و مقدار close آن را true میکنیم.

Solidity : میزان چگال بودن یک شکل را ارزیابی میکند.

$$Solidity = \frac{Area}{ConvexArea}$$

برای محاسبه محیط مانند قسمت قبل استفاده میکنیم. برای محاسبه ی مقدار convexArea ابتدا convex را با استفاده از ()convexHull و ورودی کانتور های تصویر محاسبه کرده و محیط آن را با استفاده از تابعcontourArea

بدست می آوریم.

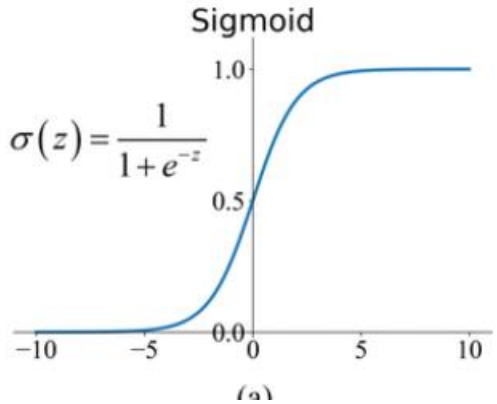
از تابع distance_criteria برای محاسبه فاصله بین دو بردار ویژگی را استفاده می شود. این دو بردار در x و y قرار دارند.

در ادامه برای ساختن features به ازای هر contour مقدار دو descriptor بالا که تعریف کردیم را محاسبه میکنیم.

.3

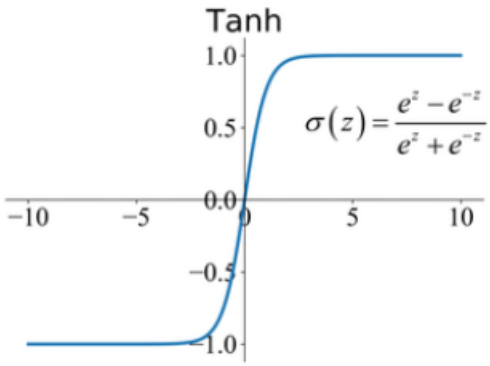
توابع فعال سازی برای جلوگیری از خطی شدن استفاده می شوند. یعنی دیتا هر بار که در نود میرود توابع خطی روی آن اعمال شده و نتیجه ی لایه ها خطی میشود و مانند این است که یک لایه ی خطی داشته ایم. بنابراین نیاز به یک تابع فعال سازی غیر خطی داریم تا از این اتفاق جلوگیری شود.

تابع sigmoid:



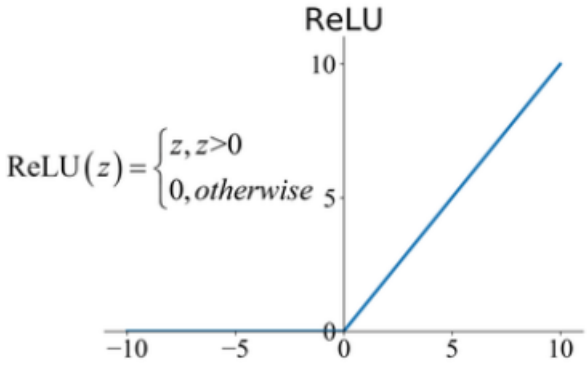
این الگوریتم یک مقدار بین ۰ و ۱ را تولید می‌کند که برای مسائل دسته‌بندی دودویی مفید است، زیرا می‌توان یک مقدار آستانه را تعیین کرد. بالاتر از این مقدار آستانه، خروجی ممکن است به عنوان ۱ در نظر گرفته شود و پایین‌تر از آن مقدار، خروجی صفر خواهد بود. این باعث می‌شود که الگوریتم LBP یک انتخاب محبوب برای گره خروجی نهایی یک شبکه باشد. همچنین درممکن است با مشکل gradient vanishing روبرو شود که موجب کاهش سرعت آموزش می شود.

تابع tanh :



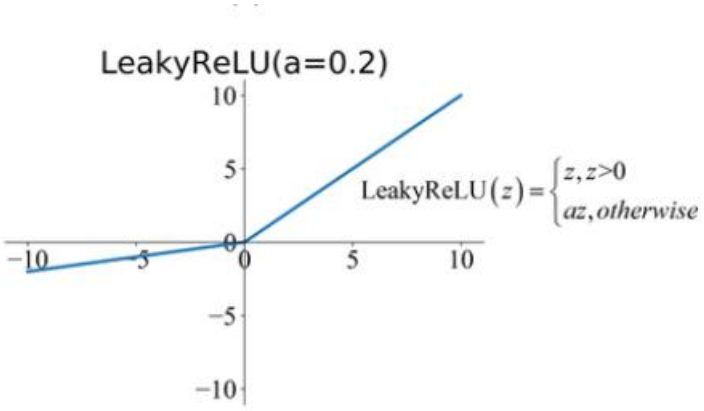
محدوده خروجی تابع تانژانت (tanh) در بازه (−1، 1) قرار دارد و رفتار مشابهی با تابع sigmoid دارد. اما تفاوت اصلی این است که تابع تانژانت مقادیر ورودی را به ۱ و −۱ فشرده می‌کند که باعث تمایز بیشتری در مقادیر مثبت و منفی به جای ۱ و ۰ کردن مقادیر ورودی.

تابع ReLU :



به دلیل خروجی صفر برای مقادیر منفی، مشکل gradient vanishing در بعضی شرایط حل می شود و به سرعت آموزش شبکه کمک می کند. همچنین برای ایجاد خاصیت فیلتراسیون در شبکه بخش های خاموش و روشن ایجاد می کند.

تابع PReLU :



درست است که تابع Leaky ReLU ، برخلاف تابع ReLU ، ورودی‌های منفی را به صفر تبدیل نمی‌کند. به جای آن، ورودی منفی را با یک مقدار کوچک (مانند ۰٫۰۲) ضرب می‌کند که این مقدار قابل آموزش است و ورودی مثبت را به همان شکل باقی می‌گذارد و باعث تطبیق بهتر با داده های ورود ی می شود. اما استفاده از تابع Leaky ReLU تقریباً بهبود قابل توجهی در دقت مدل‌های ما نشان نداده است.

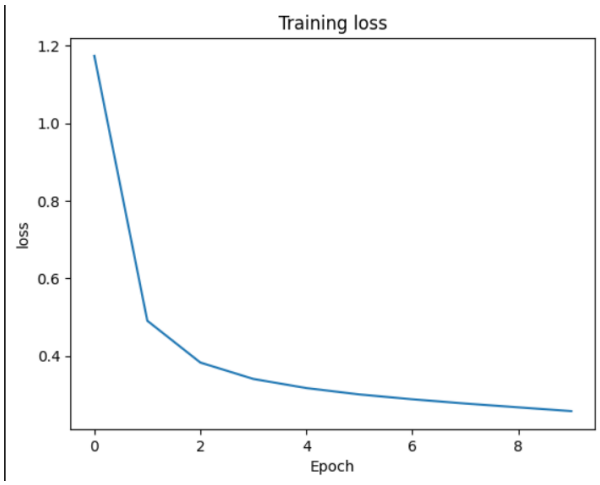
استفاده از هر کدام از این توابع با توجه به ساختار شبکه عصبی امکان پذیر است و هر کدام برای یک کار خاص استفاده میشود. sigmoid و Tanh توابعی نرمالیزه شده و خروجی‌های آن ها بازه مشخص دارند. اما relu و prelu توابعی با خروجی‌های غیرخطی هستند که برای شبکه‌های عمیق و پیچیده معمولاً عملکرد بهتری دارند. به علاوه، prelu با داشتن پارامتر قابل آموزش، با مقادیر ورودی نزدیکی و هماهنگی بیشتری دارند.

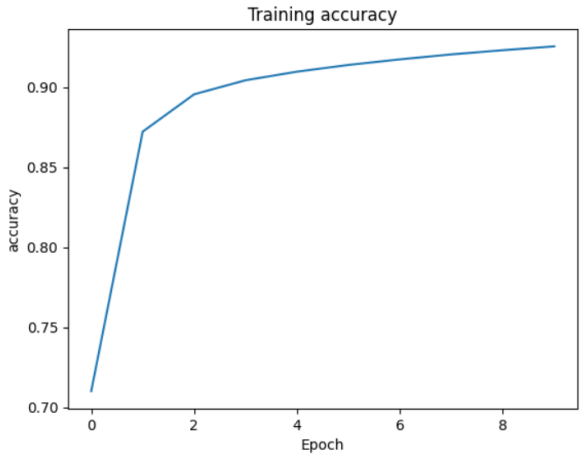
.4

در ابتدا با استفاده از keras.datasets.mnist.load_data() دیتای اعداد را لود میکنیم. بعد ابعاد آرایه های دیتا را نشان میدهیم که 60000 دیتا برای آموزش و 10000 دیتا برای تست داریم. که هر کدام یک عکس 28 در 28 هستند و هر کدام یک لیبل دارند. برای نرمالایز کردن دیتا هر دو دسته داده را تقسیم بر 255 میکنیم تا مقادیر آن بین 0 تا 1 باشد.

برای مدل sequential لایه ی input و بعد لایه ی flatten برای اینکه همه ی ورودی یک سطح بشود و بعد هم دولایه ی dence میگذاریم که فعال ساز آن ReLU است و در لایه آخر خروجی 10 تا میشود تا logit های خروجی مقادیر احتمال قرار گیری عکس در یکی از 10 دسته را نشان دهد. سپس مدل را کامپایل کرده با نرمالایزرadam و بعد هم آن را fit میکنیم در epoch 10.

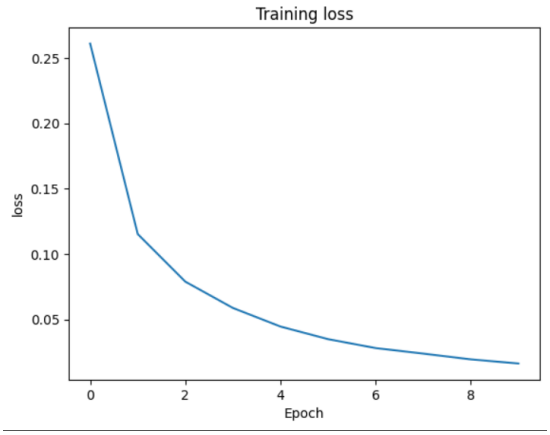
در دو سلول بعد هم مقدار loss و accuracy را برای داده train نشان میدهیم.

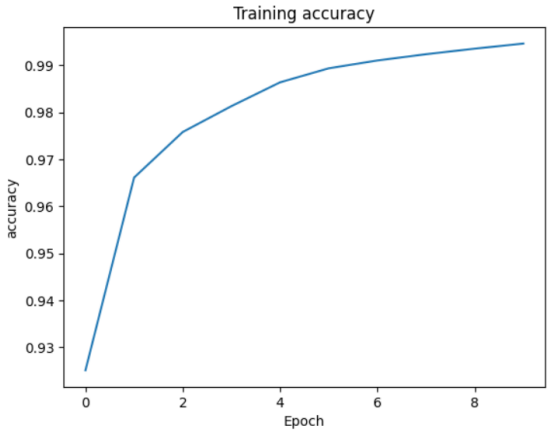




در ادامه evaluate و predict میکنیم و لایه ی softmax را میزنیم.

برای مدل functional هم همین کار هارا میکنیم با این تفاوت که مدل را به صورت گراف و functional تعریف میکنیم. خروجی:





هر شبکه‌ای را که به صورت Functional قابل پیاده‌سازی کنیم، نمی‌توان به صورت Sequential پیاده‌سازی کرد. زیرا ساختار و روابط میان لایه‌ها در دو روش Sequential و Functional متفاوت است. در روش Sequential، شبکه به صورت خطی ساختاردهی می‌شود، به این معنی که هر لایه جدید به صورت مستقیم به لایه قبلی متصل می‌شود. اما در روش Functional، شبکه به صورت گرافی ساختاردهی می‌شود، به این معنی که لایه‌ها و روابط میان آن‌ها به صورت اعمال توابع ریاضی و محاسباتی مشخص می‌شود. بنابراین، اگر یک شبکه به صورت Functional پیاده‌سازی شود، ممکن است دارای روابط و ساختاری باشد که در روش Sequential قابل پیاده‌سازی نباشد.

.6

الف) با توجه به فرمول روبه رو داریم :

$$W_2 = (W_1 - F + 2P)/S + 1$$
$$H_2 = (H_1 - F + 2P)/S + 1$$
$$D_2 = D_1$$

W2 = (7 – 7 + 2*0) / 1 + 1 = 1

H2 = 1

D2 = 3

ب)

خروجی 5 در 5 در 3 است : مرحله اول

(7 – 3 + 2*0) / 1 + 1 = 5

خروجی 3 در 3 در 3 است : مرحله دوم

(5 – 3 + 2*0) / 1 + 1 = 3

خروجی 1 در 1 در 3 است : مرحله سوم

(3 – 3 + 2*0) / 1 + 1 = 1

ج) در حالتی که از یک کرنل ۷ در ۷ بر روی یک تصویر سه کاناله استفاده می‌کنیم، برای هر کرنل ۷ در ۷ نیاز به ۱۵۰ متغیر (وزن) داریم. این شامل ۱۲۳ وزن برای ورودی‌ها (پیکسل‌ها) و ۳ وزن برای bias است.

برای کرنل 7 در 7 نیاز به 123 وزن برای ورودی ها و 3 تا برای bias داریم. برای کرنل 3 در 3 به 27 وزن برای ورودی و 3 وزن برای bias داریم. بنابراین در کل 30 وزن برای هر کرنل و در کل 90 وزن برای کل کرنل ها داریم.

بنابراین سه کرنل ۳ در ۳ چون تعداد پارامترهای کمتری دارد و محاسبات آن کمتر است بهتر است. با استفاده از کرنل های کوچیکتر روابط غیرخطی بیشتری کشف میشود و باعث بهبود عملکرد مدل می شود. بعلاوه عمق بیشتر مدل، باعث پیدا کردن ویژگی‌های پیچیده‌تر می‌شود که کیفیت را بالا میبرد.