به نام خدا

مبانی بینایی کامپیوتر

دکتر محمدی

تمرین پنج

علی عطاریان – ۹۹۵۲۱۴۵۱

سوال ۱) الف)

برای محاسبات به padding نیاز داریم اما در عمل می‌توان پدینگ را border_constant=0 قرار داد.

سوال ۱) ب)

قبل از LBP:

0	0	0	0	0	0	0	0
0	10	10	10	250	250	250	0
0	10	10	10	250	250	250	0
0	10	10	10	250	250	250	0
0	10	10	10	250	250	250	0
0	10	10	10	250	250	250	0
0	0	0	0	0	0	0	0

بعد از LBP:

3	5	5	3	5	3
5	9	9	5	9	5
5	9	9	5	9	5
5	9	9	5	9	5
3	5	5	3	5	3

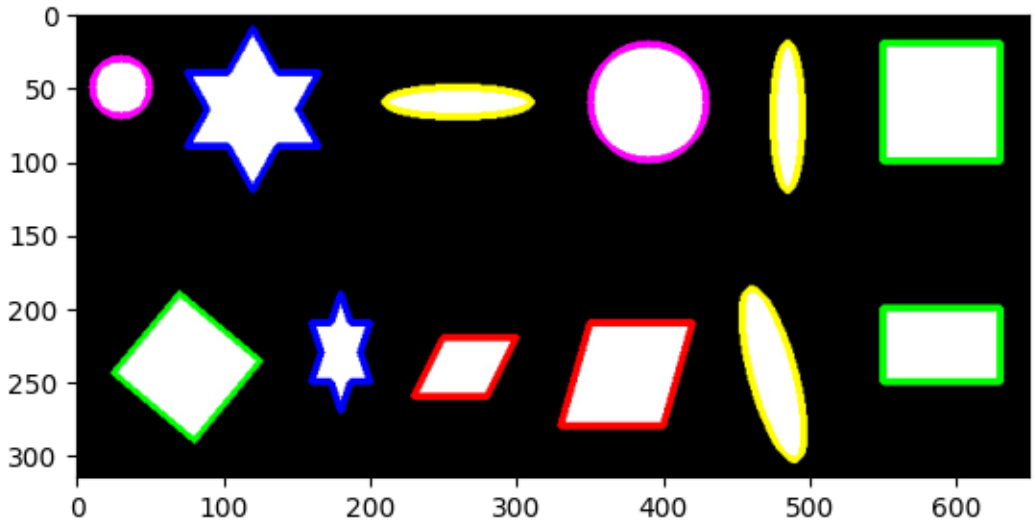
سوال ۲)

در این سوال از دو توصیف‌کننده solidity و compactness استفاده کردیم.

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

برای محاسبه اختلاف ویژگی‌ها نیز از فرمول مجموع "اختلاف به توان دو" ها استفاده می‌کنیم.

در ابتدا تنها تفاوت بین شکل ستاره و باقی چندضلعی ها را می‌یابد که یعنی solidity درست کار می‌کند و تفاوت بین شکل محدب و شکل مقعر را می‌یابد، حال برای اینکه تاثیر compactness مشهود شود آن را به علاوه یک می‌کنیم که از بازه ۰تا۱ خارج شود سپس ضربدر ۴ می‌کنیم تا تفاوت بین مقدارها مشخص باشد. پس از اعمال این تغییرات خروجی صحیح می‌شود.



سوال ۳)

برای دلایل استفاده از توابع فعال‌ساز می‌توان به موارد زیر اشاره کرد:

۱) غیرخطی بودن: باعث می‌شود شبکه بتواند الگوهای غیرخطی بین ورودی و خروجی را تشخیص دهد، در واقع بدون تابع فعال‌ساز اصلا تعدد لایه‌ها معنی ندارد چون تمام لایه ها را می‌توان با ضرب‌کردن وزن‌های متناظرشان در یکدیگر به یک لایه خطی تبدیل کرد.

۲) کنترل کردن خروجی: برای مثال تابع سیگموید خروجی را بین ۰تا۱ نرمالایز می‌کند که برای بیان احتمال و مسائل کلاس‌بندی کاربرد دارد یا تابع tanh خروجی را در بازه -۱تا۱ قرار می‌دهد که برای برخی محاسبات دیگر کاربرد دارد.

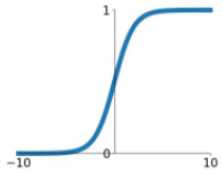
۳) بهینگی محاسباتی: توابعی مانند relu برای محاسبه بسیار آسان هستند و کار را برای لایه‌های بعدی نیز بهینه می‌کنند.

۴) کمک در بهبود وزن‌ها: در هنگام backward propagation از لایه خروجی به لایه ورودی وزن‌ها در هر لایه ضربدر مشتق تابع فعال‌ساز می‌شوند. اگر مشتق تابع مدنظر بسیار کوچک باشد، شبکه کند یاد می‌گیرد و اگر مشتق بسیار بزرگ باشد، شبکه به سمت تصادفی شدن می‌رود.
توابعی مانند relu مشتق مناسبی برای اینکار دارند.

برای مقایسه توابع می‌توان نکات زیر را مطرح کرد:

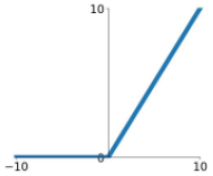
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



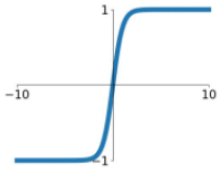
ReLU

$$\max(0, x)$$



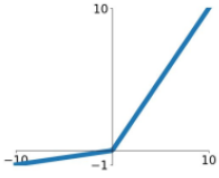
tanh

$$\tanh(x)$$



Leaky ReLU

$$\max(0.1x, x)$$



سیگموید: این تابع خروجی را به بازه ۰تا۱ مپ می‌کند که مناسب مسائل کلاس‌بندی دودویی می‌باشد اما مشکل آن این است که در ورودی‌های بسیار بزرگ یا بسیار کوچک مشتق آن نزدیک صفر است و این فرایند یادگیری را کند می‌کند

تانژانت هایپربولیک: مشابه تابع سیگموید است اما نسبت به نقطه صفر قرینه است و ورودی را به بازه -۱تا۱ مپ می‌کند. توانایی در مپ کردن به اعداد منفی گاهی برای ما مطلوب است. اما همچنان مشکل vanishing gradient مطرح شده در تابع سیگموید را دارد.

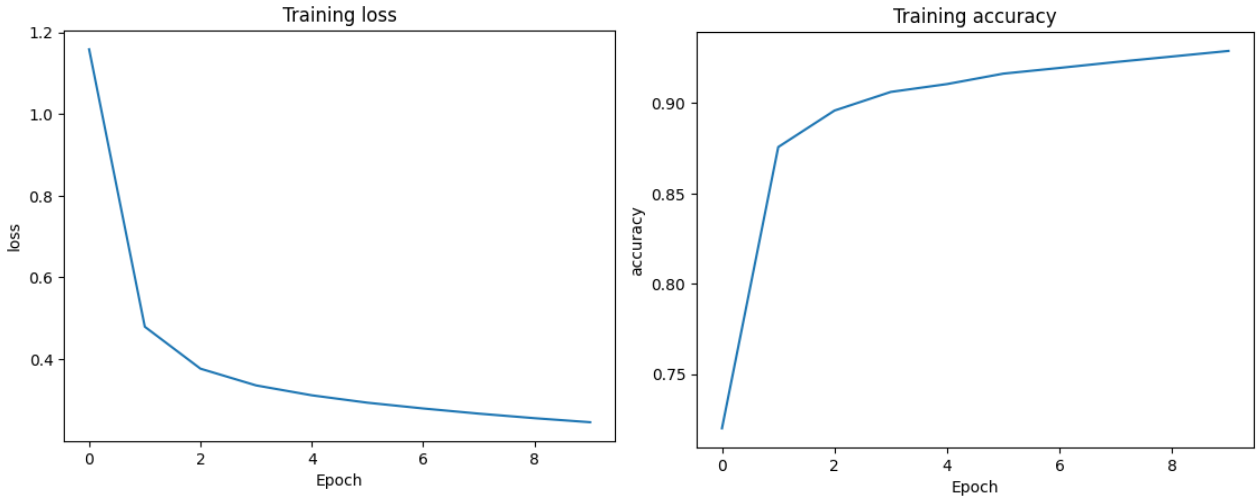
ReLu: اگر ورودی مثبت بود، خودش و اگر منفی بود صفر برمی‌گرداند. مشکل یادگیری کند دو تابع قبلی را ندارد و محاسبه مشتق آسانی دارد اما چون خروجی در مقادیر منفی صفر است باعث ایجاد نورون‌های مرده می‌شود که در واقع باعث کاهش ظرفیت یادگیری شبکه می‌شود.

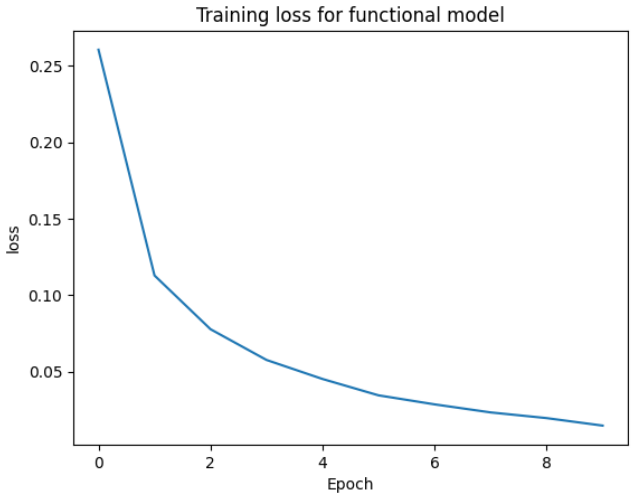
PReLu: مشابه رلو است اما یک پارامتر قابل‌یادگیری معرفی می‌کند که مقدار شیب تابع به ازای مقادیر ورودی منفی را نشان می‌دهد (اگر این مقدار را یک عدد مثبت ثابت قرار دهیم Leaky ReLu به دست می‌آید). این تابع دیگر خروجی صفر نمی‌دهد و مشکل نورون مرده تابع قبلی را برطرف می‌کند. با اینحال به علت اضافه شدن پارامتر قابل‌یادگیری اضافه به شبکه باعث پیچیدگی محاسباتی هنگام یادگیری می‌شود.

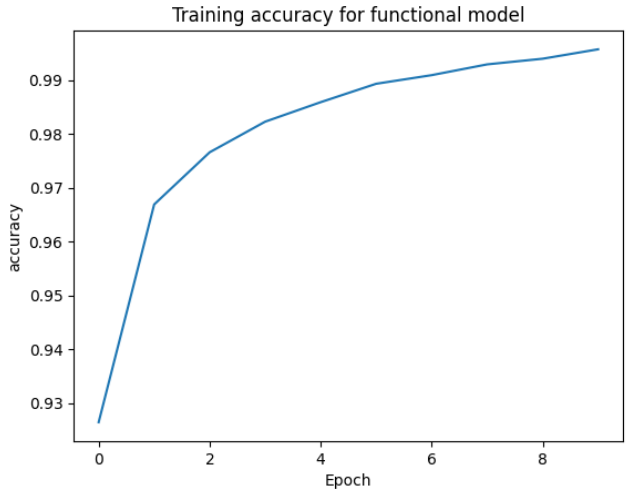
در کل از سیگموید و tanh هنگامی استفاده می‌شود که خروجی در بازه مشخص می‌خواهیم (احتمال یا کلاس‌بندی) و از ReLu و مشتقات آن در لایه‌های پنهان استفاده می‌شود که مشکل vanishing gradient را حل می‌کند و به همگرایی و تنکی شبکه کمک می‌کند.

سوال ۴)

توضیحات لازم در نوتبوک مربوطه موجود است. خروجی به شرح زیر می‌باشد:







evaluate (and analyze)

```
test_loss, test_acc = seq_model.evaluate(x_test,  y_test, verbose=2)

print(f'\nTest accuracy: {test_acc} Test loss: {test_loss}')
```

```
313/313 - 1s - loss: 0.2422 - accuracy: 0.9295 - 535ms/epoch - 2ms/step

Test accuracy: 0.9294999837875366 Test loss: 0.24218742549419403
```

analyze the result

test loss=0.2421 and train loss=0.2457 are so close so we can say it is a good model and it doesn't overfit.

evaluate (and analyze)

```
test_loss, test_acc = model.evaluate(x_test,  y_test, verbose=2)

print(f'\nTest accuracy: {test_acc} Test loss: {test_loss}')
```

```
313/313 - 1s - loss: 0.0891 - accuracy: 0.9770 - 764ms/epoch - 2ms/step

Test accuracy: 0.9769999980926514 Test loss: 0.08909997344017029
```

analyze the result

even though test_loss=0.0890 is almost 8 times bigger than train_loss=0.0148 but they are very close to each other and we can say it is a good model and this isn't a case of overfit.

سوال ۵)

خیر زیرا قابلیت‌هایی در حالت functional وجود دارد و قابل پیاده‌سازی است که در حالت sequential به آن دسترسی نداریم. برخی موارد به شرح زیر است:

۱) مدل می‌تواند ورودی‌ها و خروجی‌ها مختلف و متفاوت داشته باشد.

۲) هر کدام از لایه‌ها می‌تواند ورودی‌ها و خروجی‌های متعدد داشته باشد.

۳) به اشتراک گذاشتن لایه ها امکان‌پذیر است.

۴) داشتن توپولوژی غیرخطی امکان‌پذیر است.

سوال ۶) الف)

خروجی ابعاد (1,1,3) خواهد داشت زیرا یک پنچره ۷*۷ کاملا تصویر را پوشش می‌دهد و یک عمل کانوولوشن انجام می‌شود (در ۳ کانال).

سوال ۶) ب)

باز هم خروجی ابعاد (1,1,3) خواهد داشت زیرا تصویر پس از لایه(مرحله) اول ۵*۵ و پس از لایه دوم ۳*۳ خواهد شد بنابراین در آخرین مرحله کانوولوشن، ابعاد تصویر و ابعاد فیلتر برابر است پس خروجی یک عدد است (۳ کانال متفاوت).

سوال ۶) ج)

در حالت اول 7*7*3+1=148 پارامتر یاد گرفته می‌شود اما در حالت دوم در هر مرحله 3*3*3+1=28 پارامتر آموزش داده می‌شود که جمعا می‌شود ۸۴ پارامتر. با آنکه تعداد پارامترها در حالت اول بیشتر است اما ویژگی‌های سطحی‌تری را استخراج می‌کنند زیرا تمام این پارامترها مربوط به یک معادله خطی هستند در حالی که در حالت دوم پس از هر مرحله یک تابع غیرخطی اعمال می‌شود و پارامترها مربوط به معادله غیرخطی هستند پس ویژگی‌های بهتری و عمیق‌تری استخراج می‌شود.