بسمه تعالی

دانشکده مهندسی کامپیوتر

یادگیری عمیق

نام استاد: دکتر محمدی

تمرین نهم

آرمان حیدری

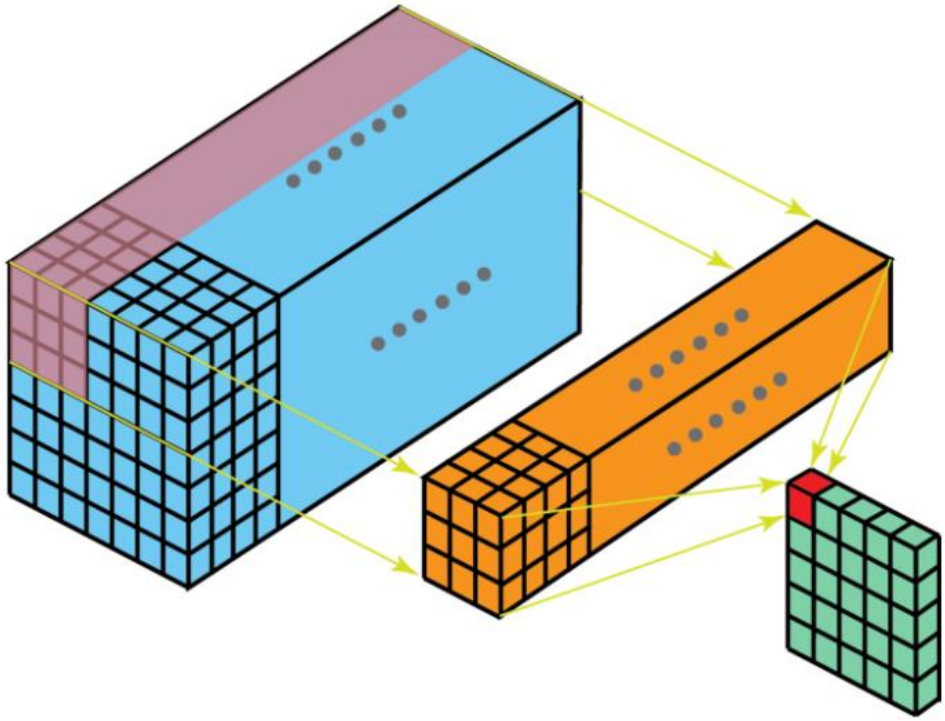
شماره دانشجویی: ۹۷۵۲۱۲۵۲

آذر ۱۴۰۰

۱. پاسخ سوال اول

الف) می‌توانیم فرآیند کانوولوشن دو بعدی را به عنوان لغزش یک ماتریس فیلتر سه بعدی در لایه ورودی در نظر بگیریم. توجه داشته باشید که لایه ورودی و فیلتر دارای عمق یکسانی هستند (شماره کانال = شماره kernel). فیلتر سه بعدی فقط در دو جهت، ارتفاع و عرض تصویر حرکت می کند (به همین دلیل است که چنین عملیاتی به عنوان کانوولوشن دو بعدی نامیده می شود، اگرچه از یک فیلتر سه بعدی برای پردازش داده های حجمی سه بعدی استفاده می شود). در هر موقعیت لغزشی، ضرب و جمع را بر حسب عنصر انجام می دهیم که منجر به یک عدد می شود. به طور کلی، ما یک کانال خروجی واحد دریافت می کنیم.

اکنون می‌توانیم ببینیم که چگونه می‌توان بین لایه‌هایی با عمق‌های مختلف انتقال ایجاد کرد. فرض کنید لایه ورودی دارای کانال های Din است و ما می خواهیم لایه خروجی دارای کانال های Dout باشد. کاری که باید انجام دهیم این است که فقط فیلترهای Dout را روی لایه ورودی اعمال کنیم. هر فیلتر دارای هسته din است. هر فیلتر یک کانال خروجی را ارائه می دهد. پس از اعمال فیلترهای Dout، کانال‌های Dout داریم که می‌توان آن‌ها را در کنار هم قرار داد تا لایه خروجی را تشکیل دهند. شکل زیر این موضوع را به خوبی نمایش می دهد:



از جمله کاربرد های آن میتوان به پردازش سیگنال ها اشاره کرد که اگر پاسخ ضربه ای سیستم را بدانیم میتوانیم خروجی را حدس بزنیم. همچنین پرکاربرد ترین آن استخراج ویژگی از عکس ها که دو بعدی ۱ کاناله یا ۳ کاناله هستند. برای یافتن الگوهای موجود در تصویر مثل لبه ها، رنگ ها و ... مناسب است. عاملی که با عث محبوبیت آن شده این است که این کانولوشن میتواند ویژگی های مکانی را به خوبی استخراج کند. استفاده از آن شبکه را در کارهای مربوط به تصاویر Robust می کند. و به طور کلی classification images، Segmentation، Object Detection

همچنین convolution های سه بعدی وجود دارد. آنها تعمیم convolution دو بعدی هستند. در اینجا در کانولوشن سه بعدی، عمق فیلتر کمتر از عمق لایه ورودی است (اندازه هسته < اندازه کانال). در نتیجه، فیلتر سه بعدی می تواند در هر سه جهت (ارتفاع، عرض، کانال تصویر) حرکت کند. در هر موقعیت، ضرب و جمع بر حسب عنصر یک عدد را ارائه می دهد. از آنجایی که فیلتر از یک فضای سه بعدی عبور می کند، اعداد خروجی نیز در یک فضای سه بعدی مرتب می شوند. سپس خروجی یک داده سه بعدی است.

همانند کانولوشن های دوبعدی که روابط فضایی اشیاء را در یک دامنه دو بعدی رمزگذاری می کنند، کانولوشن های سه بعدی نیز می توانند روابط فضایی اشیاء در فضای سه بعدی را توصیف کنند. چنین رابطه سه بعدی برای برخی از کاربردها مهم است، مانند بخش بندی های سه بعدی، بازسازی های تخیل زیست پزشکی، CT و MRI که در آن اشیایی مانند رگ های خونی در فضای سه بعدی پیچ و تاب می خورند.

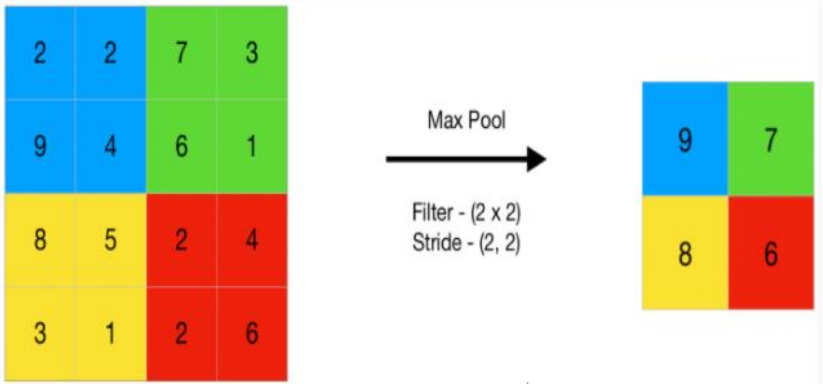
ب) زیرا لایه های کانوولوشنی مهمترین استفاده شان در شبکه های عمیق مبتنی بر عکس ها می باشد. و در مواردی مانند تصاویر یا حتی ویدئو ها، مولفه سوم ورودی یا همان اندازه کانال خودش نشان دهنده ویژگی های مختلف است. با یک مثال این موضوع را بیان میکنیم. هر پیکسل عکسی که RGB است، کانال ورودی اندازه ۳ را دارد که این اعداد در کنار هم نشاندهنده ویژگی های آن پیکسل هستند.

هدف ما هم از لایه کانوولوشن استخراج ویژگی هایی معنا دار در سراسر عکس می باشد. پس کانوولوشنی که مولفه های کانال را در کنار هم برای هر پیکسل بررسی کند مطلوب است. که چنین حالتی در convolution 2D وجود دارد. اگر میخواستیم ۳ بعدی استفاده کنیم، یعنی فیلتر های کانوولوشنی را مثلا فقط روی اعداد آبی و

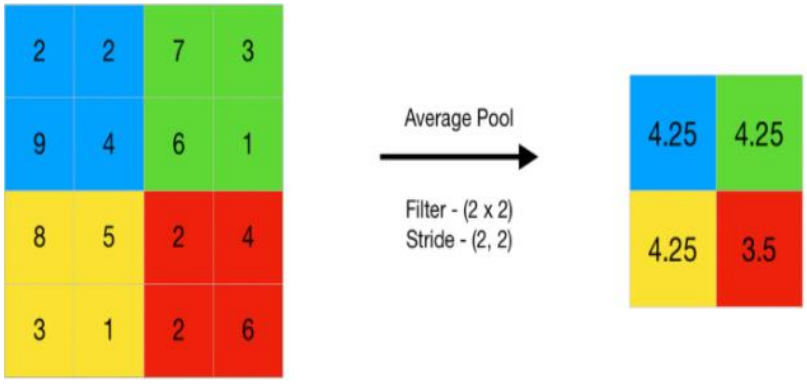
قرمز پیکسل ها (و نه سبز ها) اعمال میکردیم، که در اکثر مواقع چنین امکانی را نیاز نداریم. پس از مربعی استفاده میکنیم.

ج) لایه های pooling برای کاهش ابعاد نقشه های ویژگی استفاده می شود. بنابراین، تعداد پارامترهای یادگیری و میزان محاسبات انجام شده در شبکه را کاهش می دهد. لایه pooling ویژگی های موجود در یک منطقه از نقشه ویژگی ایجاد شده توسط یک لایه کانولوشن را خلاصه می کند. بنابراین، عملیات بیشتر بر روی ویژگی‌های خلاصه شده به‌جای ویژگی‌های دقیقاً موقعیت‌یافته تولید شده توسط لایه کانوولوشن انجام می‌شود. این باعث می شود که مدل نسبت به تغییرات در موقعیت ویژگی ها در تصویر ورودی قوی تر باشد. همچنین این لایه ها پارامتر قابل آموزش ندارند و کمتر باعث پیچیده شدن شبکه می شوند. انواع این لایه:

- Max pooling: مقدار ماکزیمم ناحیه را انتخاب می کند. بنابراین، خروجی پس از لایه max-pooling یک نقشه ویژگی خواهد بود که شامل برجسته ترین ویژگی های نقشه ویژگی قبلی است.



- Average pooling: میانگین عناصر موجود در منطقه نقشه ویژگی تحت پوشش فیلتر را محاسبه می کند. بنابراین، در حالی که max pooling برجسته ترین ویژگی را در یک قسمت خاص از نقشه ویژگی می دهد، average pooling میانگین ویژگی های موجود در یک بخش را ارائه می دهد.



- Global pooling: هر کانال در نقشه ویژگی را به یک مقدار کاهش می دهد. بنابراین، یک نقشه ویژگی n_h x n_w x n_c به نقشه ویژگی $1*1*n_c$کاهش می یابد. این معادل استفاده از فیلتری با ابعاد n_h x n_w یعنی ابعاد نقشه ویژگی است. می تواند به صورت global max pooling یا global average pooling باشد.
- Min pooling: درست مانند max pooling است فقط به جای بزرگترین عدد، کوچکترین را برمی‌گزینیم. زمانی مفید است که پس‌زمینه تصویر تاریک است و ما فقط به پیکسل‌های روشن‌تر تصویر علاقه‌مندیم. به عنوان مثال: در مجموعه داده MNIST، ارقام به رنگ سفید و پس‌زمینه سیاه نشان داده می‌شوند. بنابراین، max pooling استفاده می شود. به طور مشابه، Min Pooling برعکس استفاده می شود. که البته کاربرد کمتری دارد.
- Adaptive pooling: در حالات قبلی، شما اساساً گام و اندازه هسته را خودتان تنظیم می‌کنید و آنها را به‌عنوان hyperparameter تنظیم می‌کنید. اگر اندازه ورودی خود را تغییر دادید، باید آنها را دوباره پیکربندی کنید. از طرف دیگر در Adaptive Pooling، به جای آن اندازه خروجی را مشخص می کنیم. و گام و اندازه هسته به طور خودکار برای انطباق با نیازها انتخاب می شوند. خودش میتواند average adaptive pooling یا max adaptive pooling باشد. که از نظر نتیجه فرق خاصی با آن ها ندارد.

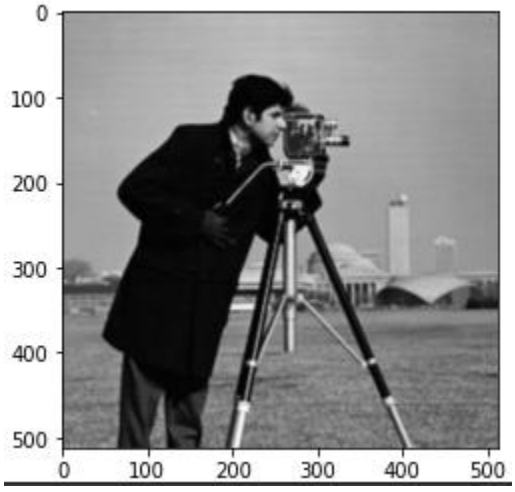
منابع) towardsdatascience، GeekForGeeks، ai.plainenglish، stackoverflow

۲. پاسخ سوال دوم

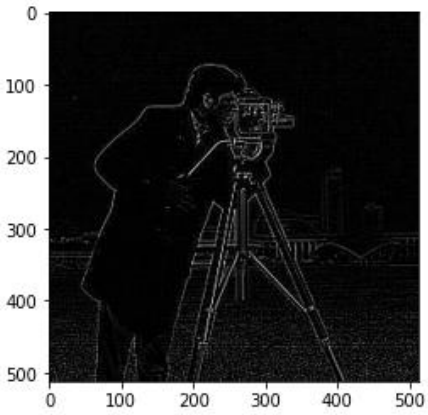
کد مربوط به این قسمت در hw9.ipynb بخش question 2 زده شده است. (لینک گوگل کولب)

از چپ به راست آن ها را شماره گذاری کرده ام (kernel1،kernel2، kernel3، kernel4). توضیحات و خروجی هر کدام به این صورت بوده است:

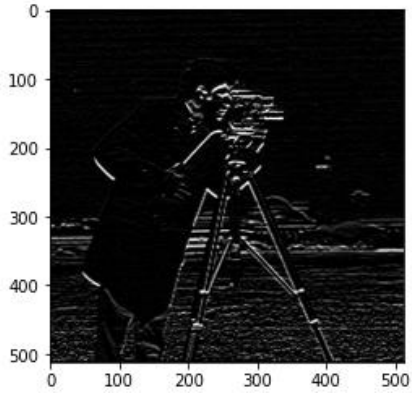
- Kernel1: این فیلتر برای blur کردن تصویر استفاده میشود. معمولا از این فیلتر بعنوان smooth کننده تصویر استفاده میشود. در واقع این فیلتر برای هرپیکسل، مقدار میانگین پیکسل های اطراف را (به اندازه سایز فیلتر) را محاسبه میکند و جایگزاری میکند. به این نوع تار کردن average smoothing نیز میگویند. هر چقدر مقدار مخرج کسر بیشتر باشد یعنی تاثیر خانه ها کمتر میشود و تصویر smooth تر میشود.



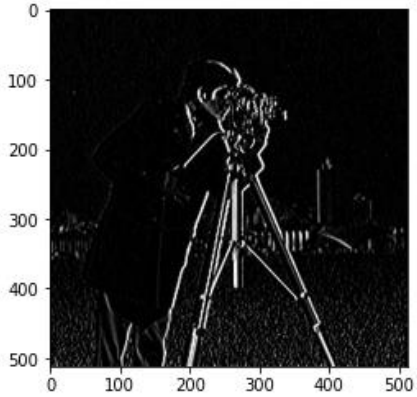
- Kernel2: یک لبه یاب یا Edge Detegtor است. از ورودی مشتق دوم میگیرد تا نرخ تغییرات مشتق اول را نشان دهد. بنابراین میتواند Edge را تشخیص دهد. کار آن طبق سایت‌های مختلف، تشخیص rapid intensity change است. از خروجی هم مشخص است که لبه ها پیدا شده است.



- Kernel3: این فیلتر soble در جهت محور عمودی y است. و درواقع کارش تشخیص تغییرات افقی گرادیان عکس است. با اینکار میتوانیم لبه های افقی را پیدا کنیم.



- Kernel4: این فیلتر soble در جهت محور افقی x است. و درواقع کارش تشخیص تغییرات عمودی گرادیان عکس است. با اینکار میتوانیم لبه های عمودی را پیدا کنیم.



۳. پاسخ سوال سوم

الف) از این framework برای یافتن مقادیر مناسب hyperparameter های شبکه استفاده میکنیم. که موضوعی زمان بر و سخت محسوب می شود. مقیاس پذیر است و می تواند بهترین عملکرد شبکه را در بازه هایی از مقادیر ابرپارامتر ها پیدا کند و به ما بگوید که با چه مقادیری بهترین نتیجه به دست آمده است. درواقع الگوریتم های جستجوی مختلفی دارد. توابع مختلف آن که می توانیم برای تعریف بازه یا بودن و نبودن هایپرپارامتر های مختلف از آن استفاده کنیم:

Boolean, Choice, Int, Fixed, Float

با تعریف کردن مقادیر min،max، sampling، step برای این توابع، هنگام جستجوی بهترین الگوریتم با مقادیر مختلف شبکه را اعمال میکند. (با تعداد epoch معینی که به آن میدهیم و معمولا نباید خیلی زیاد باشد چون زمانبر است)

ب) در قسمت قبل گفتیم که الگوریتم های مختلفی برای جستجوی بهترین مقادیر ابرپارامترها وجود دارد. این موضوع و این که ما شبکه را با چه کتابخانه ای ایجاد کردیم میتوانند در انتخاب tuner مناسب اثرگذار باشند. Tuner های این framework عبارتند از:

- Base tuner: کارش ساختن، آموزش دادن و سنجیدن دقت مدل است. درواقع همه ی tuner ها از آن ارث بری میکنند.
- Grid search: تمامی حالات ممکن از ابرپارامتر های داده شده را امتحان می کند. که بسیار بسیار زمانبر است چون با اضافه کردن هر ابرپارامتر به قضیه این مقدار به صورت توانی رشد میکند! به همین دلیل زیاد استفاده نمی شود.
- Random search: به صورت تصادفی حالاتی را امتحان میکند و بهترین را پیدا میکند. احتمالا دقیقا بهترین جواب را نمیابد اما اگر به اندازه کافی اجرا شود جوابی مطلوب خواهد داشت.

- Hyperband: این روش مشکل اصلی روش تصادفی که امتحان کردن زیادی و وقت گذاشتن بر روی حالات نامناسب است را رفع میکند. به این صورت که پس از انتخاب تصادفی یک مجموعه از هایپرپارامت رها به جای اینکه شبکه را به صورت کامل آموزش دهد و در ادامه ارزیابی کند، مدل را تنها برای چند ایپاک آموزش می دهد و بهترین نتایج را در این چند ایپاک به مرحله بعدی آموزش می فرستد. این کار مکررا انجام می شود تا در نهایت آموزش کامل بر روی نمایندگان نهایی صورت گیرد.
- Bayesian optimization: این روش به کل سعی میکند مشکل تصادفی انتخاب شدن را حل کند. در این روش به جای اینکه تمامی حالات به صورت رندوم باشند، چند حالت ابتدایی را رندوم در نظر می گیرد. سپس با توجه به نتایج این حالات ابتدایی، بهترین حالت ممکن بعدی را می سازد. در و اقع در این روش از تاریخچه ی حالتهای قبلی برای ساختن حالت جدید استفاده میشود. این کار تا زمانی ادامه مییابد که یا به بهترین جواب برسیم یا به حداکثر تعداد آزمایش ها برسیم.
- Sklearn tuner: مخصوص مدل های ساخته شده با Sci-kit learn است.

در این سوال از روش hyperband استفاده کرده ام زیرا زمان کمتری برای اجرا نسبت به grid search میبرد و منابع gpu و زمان محدودی داشتم. اگرچه ممکن است بهترین جواب را ندهد اما جوابی نزدیک به بهترین را می دهد و از Random هم بهینه تر از منابع استفاده میکند. مدل را با keras ساخته ام پس sklearn tuner هم انتخاب خوبی نیست. Bayesian هم استفاده نکرده ام چون استفاده از این روش هنگامی که ابرپارامتر ها بازه خیلی بزرگی دارند مناسب است که در این مسئله چنین نبود.

ج) کد مربوط به این قسمت در hw9.ipynb بخش question 3 زده شده است. (لینک گوگل کولب)

مدلی طراحی میکنیم و hyperparameter های آن را با استفاده از tuner، حالات مختلفی تعیین میکنیم. به طور کلی با توجه به جدول صورت سوال چنین حالاتی در نظر میگیریم:

```
Search space summary
Default search space size: 6
conv_layers (Int)
{'default': None, 'conditions': [], 'min_value': 1, 'max_value': 3, 'step': 1, 'sampling': None}
dense_layers (Int)
{'default': None, 'conditions': [], 'min_value': 1, 'max_value': 4, 'step': 1, 'sampling': None}
units (Int)
{'default': None, 'conditions': [], 'min_value': 128, 'max_value': 256, 'step': 64, 'sampling': None}
filters (Int)
{'default': None, 'conditions': [], 'min_value': 8, 'max_value': 32, 'step': 8, 'sampling': None}
learning_rate (Choice)
{'default': 0.0001, 'conditions': [], 'values': [0.0001, 0.0005, 0.001], 'ordered': True}
optimizer (Choice)
{'default': 'adam', 'conditions': [], 'values': ['adam', 'sgd'], 'ordered': False}
```

تعدادی محدودیت هم خودمان طراحی کرده ایم که در اکثر مدل های عمیق رعایت می شود. بدین صورت مدل زمان کمتری برای حالات اشتباه صرف میکند و احتمالا جواب بهینه تری می دهد. از لایه dropout و maxpooling هم برای جلوگیری از overfit شدن و همچنین زیاد نشدن پارامترهای آموزشی استفاده میکنیم.

Hyperband را با ۳۰ بار تلاش اجرا میکنیم. و به چنین نتایجی برای حالت بهینه میرسد:

```
Showing 10 best trials
Objective(name='val_accuracy', direction='max')
Trial summary
Hyperparameters:
conv_layers: 2
dense_layers: 2
units: 128
filters: 24
learning_rate: 0.0005
optimizer: adam
tuner/epochs: 10
tuner/initial_epoch: 4
tuner/bracket: 2
tuner/round: 2
tuner/trial_id: fb17ea19791ceaaaaef1cd0a85512a90
Score: 0.6915000081062317
Trial summary
Hyperparameters:
conv_layers: 1
dense_layers: 4
units: 192
filters: 16
learning_rate: 0.001
optimizer: adam
```

البته ریز نتایج epoch ها را چون خیلی طولانی میشد نمایش ندادیم اما خود tuner بهترین را یافته است. و در اینجا ۱۰ مدل برتر را نمایش داده است که در عکس فقط مورد اول را آورده ام اما در نوتبوک به طور کامل قابل بررسی است. که میبینیم مطابق انتظار بهترین optimizer، adam بوده است. تعداد نورون های زیاد را بهینه ندانسته و همان ۱۲۸ که کمترین مقدار پیشنهادی بوده را بازگردانده است. تعداد لایه های dense و conv و learning rate هم در تصویر مشخص است. بهترین مدل هم با ۱۰ بار اجرا به دقت ۷۰ درصد روی داده validation رسیده است که عدد قابل قبولی است.

حالا بهترین مدل را بر این اساس میسازیم. که چنین مدلی است:

```
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 32, 32, 3)]       0

conv2d_6 (Conv2D)            (None, 32, 32, 48)        1344

conv2d_7 (Conv2D)            (None, 32, 32, 48)        20784

max_pooling2d_3 (MaxPooling  (None, 16, 16, 48)        0
2D)

dropout_6 (Dropout)          (None, 16, 16, 48)        0

conv2d_8 (Conv2D)            (None, 16, 16, 96)        41568

conv2d_9 (Conv2D)            (None, 16, 16, 96)        83040

max_pooling2d_4 (MaxPooling  (None, 8, 8, 96)          0
2D)

dropout_7 (Dropout)          (None, 8, 8, 96)          0

flatten_1 (Flatten)          (None, 6144)              0

dense_4 (Dense)              (None, 128)               786560

dropout_8 (Dropout)          (None, 128)               0

dense_5 (Dense)              (None, 128)               16512

dropout_9 (Dropout)          (None, 128)               0

dense_6 (Dense)              (None, 10)                1290

=================================================================
Total params: 951,098
Trainable params: 951,098
Non-trainable params: 0
```

آن را با ۵۰ epoch آموزش میدهیم(چون محدودیت gpu اجازه اجرای بیشتر را نمیداد). روند آموزش کاملا مشخص است و همگرا هم نشده و با ادامه به دقت بهتر هم میرسد. در نهایت دقتش را روی داده آموزشی و تست ارزیابی میکنیم:

```
1563/1563 [==============================] - 11s 7ms/step - loss: 0.2014 - accuracy: 0.9442
train loss and accuracy: [0.2013693004846573, 0.944159984588623]
313/313 [==============================] - 2s 7ms/step - loss: 0.6705 - accuracy: 0.7962
test loss and accuracy: [0.6704970598220825, 0.7961999773979187]
```

منابع: medium

۴. پاسخ سوال چهارم

برای این سوال لایه به لایه مقادیر قابل یادگیری را محاسبه میکنیم و حاصل جمع آن ها جواب مسئله است.

- برای لایه اول Convolution2D: این لایه شامل ۲۰ تا فیلتر میباشد و هر فیلتر دارای سایز 7x7x1 میباشد که مقدار ۱ بخاطر این است که ورودی تصویر با توجه به shape ورودی، ۱ کاناله میباشد.

$$20 * (7 * 7 * 1 + 1) = 1000$$

- برای لایه دوم MaxPooling2D: این لایه پارامتر قابل یادگیری ندارد.
- برای لایه سوم Convolution2D: مشابه با کانوولوشن قبلی است با این تفاوت که برای این لایه، input shape متفاوت است و ورودی سوم آن که مولفه سوم اندازه فیلترهای ما نیز هست برابر ۲۰ می باشد. چون در کانوولوشن اولی ۲۰ ویژگی پیدا کرده بودیم. پس ۱۰ تا فیلتر داریم که سایز آنها برابر 5x5x20 میباشد. Bias هم باید محاسبه کنیم:

$$10 * (5 * 5 * 20 + 1) = \ 5010$$

- برای لایه چهارم LocallyConnected2D: کارکرد این لایه مانند لایه های Conv2D می‌باشد با این تفاوت که وزن های فیلتر ها اشتراک ندارند و محاسباتش کمی فرق میکند. در این لایه ۲ فیلتر داریم که برای هر فیلتر دارای سایز 3x3x10 میباشد که مقدار ۱۰ بخاطر این است که در لایه کانوولوشن قبلی خروجی shape، (7,7,10) خواهد بود. (11-5+1=7)

مقدار ۷ های بالا به این صورت است که فیلتر های لایه سوم مربع های 5x5 بوده اند و وقتی روی ورودی حرکت کنند(عملیات convolution) مقدار سایز عکس را کمتر کرده و تبدیل به 7x7 میکنند و چون ۱۰ فیلتر هم داریم پس در نهایت خروجی لایه سوم دارای shape = 7x7x10 میباشد.

چون فیلتر ها locally connected میباشند پس وزن های آنها اشتراک ندارد. اگر ابعاد تصویر ورودی برابر height x width x channels و ابعاد فیلتر برابر a x a باشد برای یک فیلتر به اندازه فرمول زیر وزن داریم:

وزن های هر فیلتر: a * a * channels

تعداد فیلترها: (Height – a + 1) * (width – a + 1)

و به تعداد هر کدام از فیلتر های جدید نیز یک بایاس داریم که تعداد فیلتر های جدید به صورت زیر است:

$$(height - a + 1) * (width - a + 1)$$

در نهایت تعداد پارامتر های قابل یادگیری این لایه بصورت زیر میباشد:

$$2 * ((7 - 3 + 1) * (7-3+1) * 3 * 3 * 10 + (7 - 3 + 1) * (7-3+1)) = 2 * (25 * 90 + 25) = 4550$$

درواقع تعداد ۲ فیلتر خروجی را در تعداد وزن ها + تعداد bias ها ضرب کرده ایم. خروجی این لایه دارای shape = (5 ,5 , 2) است.

- لایه پنجم Flatten: این لایه پارامتر قابل یادگیری ندارد.
- لایه ششم Dense: چون در این نوع لایه، هر نورون به نورون های قبلی وصل است پس تعداد وزن ها برابر حاصلضرب نورون های لایه قبلی در نورون های این لایه است و همچنین به ازای هر یک از نورون های این لایه نیز یک بایاس داریم که میشود ۱۰ بایاس و تعداد وزن ها هم برابر زیر میباشد:

وزن : $5 * 5 * 2 * 10 = 500$

$$500 + 10 = 510$$

Total trainable parameters: 1000 + 5010 + 4550 + 510 = 11070

می توانیم برای اطمینان از این اعداد با model.summary() از کتابخانه keras هم چک کنیم که میبینیم مطابقت دارد:

کد مربوط به این قسمت در hw9.ipynb بخش question 4 زده شده است. (لینک گوگل کولب)

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 22, 22, 20)        1000

 max_pooling2d (MaxPooling2D  (None, 11, 11, 20)       0
 )

 conv2d_1 (Conv2D)           (None, 7, 7, 10)          5010

 locally_connected2d (Locall  (None, 5, 5, 2)          4550
 yConnected2D)

 flatten (Flatten)           (None, 50)                0

 dense (Dense)               (None, 10)                510

=================================================================
Total params: 11,070
Trainable params: 11,070
Non-trainable params: 0
```

منابع:

https://stackoverflow.com/questions/42786717/how-to-calculate-the-number-of-parameters-for-convolutional-neural-network

https://stackoverflow.com/questions/34393876/difference-between-local-and-dense-layers-in-cnns