به نام خدا

تمرین سوم

دستیاران : سحرسرکار، فائزه صادقی، پریسا ظفری، شایان موسوی نیا

دانشجو : سارا سادات یونسی – ۹۸۵۳۳۰۵۳

سوال ۱)

الف)

استفاده از نرخ یادگیری بسیار بالا ممکن است باعث شود که شبکه عصبی در یادگیری دچار مشکلاتی شود. برخی از این مشکلات عبارتند از:

• از دست دادن نقاط بهینه: اگر نرخ یادگیری خیلی بالا باشد، ممکن است شبکه عصبی نتواند در نقاطی که تابع اتلاف کمترین مقدار را دارد متوقف شود و از آنها عبور کند. این باعث میشود که شبکه عصبی در نقاط بهینه محلی یا جهانی قرار نگیرد و دقت کمتری داشته باشد.

• افزایش نوسانات: اگر نرخ یادگیری خیلی بالا باشد، ممکن است شبکه عصبی در هر مرحله بهینه سازی، تغییرات بزرگی در وزنها ایجاد کند. این باعث میشود که شبکه عصبی در فضای پارامترها نوسانات زیادی داشته باشد و نتواند به یک حالت پایدار برسد.

• افزایش زمان یادگیری: اگر نرخ یادگیری خیلی بالا باشد، ممکن است شبکه عصبی نتواند به یک مقدار کمینه برای تابع اتلاف همگرا شود و در نتیجه زمان یادگیری طولانیتر شود. این ممکن است باعث شود که شبکه عصبی در مواجهه با دادههای جدید عملکرد خوبی نداشته باشد.

همچنین انفجار گرادیان و اورفلو کردن وزن ها به ترتیب به دلایل مدل نمی تواند در مقدار های محلی همگرا شود و مورد دیگر هم وزن ها تغییرات بزرگی خواهند داشت که دچار مشکل می کند.

برای تشخیص این مشکلات، میتوان از روشهای زیر استفاده کرد:

• رسم نمودار تغییرات تابع اتلاف: اگر نمودار تابع اتلاف نشان دهد که شبکه عصبی در حال از دست دادن نقاط بهینه یا افزایش نوسانات است، ممکن است نرخ یادگیری خیلی بالا باشد. در این صورت، باید نرخ یادگیری را کاهش داد.

• مقایسه دقت شبکه عصبی بر روی دادههای آموزش و تست: اگر شبکه عصبی بر روی دادههای آموزش دقت بالایی داشته باشد اما بر روی دادههای تست دقت پایینی داشته باشد، ممکن است شبکه عصبی دچار بیش برازش شده باشد. این ممکن است ناشی از نرخ یادگیری خیلی بالا باشد. در این صورت، باید نرخ یادگیری را کاهش داد.

همچنین همگرا نشدن و کانورج نشد به یک نقطه اوپتیموم و اورفیت شدن مدل و طولانی شدن سرعت یادگیری از دلایل دیگری هستند.

ب)

استفاده از نرخ یادگیری بسیار پایین ممکن است باعث شود که شبکه عصبی در یادگیری دچار مشکلاتی شود. برخی از این مشکلات عبارتند از:

- کاهش سرعت همگرایی: اگر نرخ یادگیری خیلی پایین باشد، ممکن است شبکه عصبی در هر مرحله بهینه سازی، تغییرات کوچکی در وزنها ایجاد کند. این باعث میشود که شبکه عصبی برای رسیدن به یک مقدار کمینه برای تابع اتلاف زمان زیادی نیاز داشته باشد.
- گیر افتادن در حداقل محلی: اگر نرخ یادگیری خیلی پایین باشد، ممکن است شبکه عصبی در نقاطی که تابع اتلاف مقدار کمی دارد اما بهینه نیستند، متوقف شود و نتواند از آنها خارج شود. این باعث میشود که شبکه عصبی در حداقل محلی ناخواسته گیر کند و دقت کمتری داشته باشد.

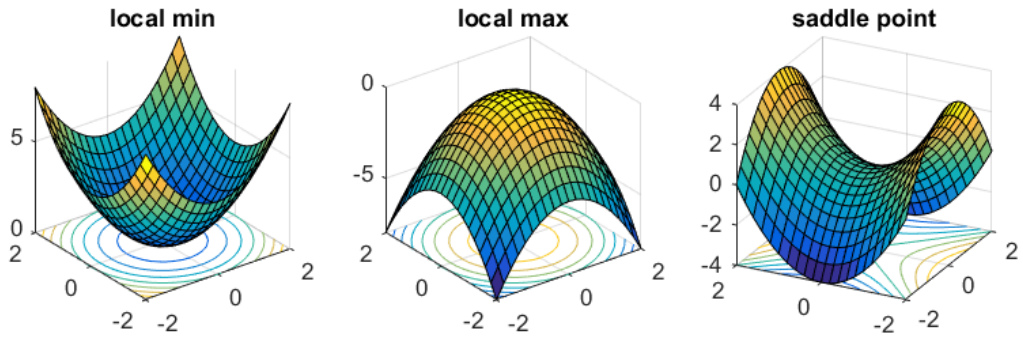
و دچار محوشدگی گرادیان شویم چون وزن ها بسیار کم تغییر می کنند و یادگیری کم می شود.

برای تشخیص این مشکلات، میتوان از روشهای زیر استفاده کرد:

- رسم نمودار تغییرات تابع اتلاف: اگر نمودار تابع اتلاف نشان دهد که شبکه عصبی در حال کاهش کندی تابع اتلاف است، اما با سرعت کم و یا در یک نقطه متوقف شده است، ممکن است نرخ یادگیری خیلی پایین باشد. در این صورت، باید نرخ یادگیری را افزایش داد.
- مقایسه دقت شبکه عصبی بر روی دادههای آموزش و تست: اگر شبکه عصبی بر روی دادههای آموزش و تست دقت مشابهی داشته باشد، اما دقت کلی کم باشد، ممکن است شبکه عصبی دچار کم برازش شده باشد. این ممکن است ناشی از نرخ یادگیری خیلی پایین باشد. در این صورت، باید نرخ یادگیری را افزایش داد.و اینکه مدل روی داده های اموزشی ارام باشد و بهبود را نبینیم یا روند اموزش بیش از حد طول بکشد .

پ)

نقطه زینی نقطهای در دامنه یک تابع است که یک نقطه سکون بوده ولی اکسترمم موضعی نیست. به عبارت دیگر، نقطهای که در آن گرادیان تابع صفر است ولی نه بیشینه و نه کمینه محلی است. نقطه زینی میتواند باعث ایجاد مشکل برای الگوریتمهای بهینهسازی گرادیان کاهشی شود، زیرا آنها را در یک نقطه غیر بهینه گیر میدهد. برای مثال، تابع z = x^2 - y^2 یک نقطه زینی در مبدا دارد.



الگوریتم Adam یک الگوریتم بهینهسازی است که میتوان از آن به جای روش گرادیان کاهشی تصادفی کلاسیک برای بهروزرسانی وزنهای شبکه بر اساس تکرار در دادههای آموزشی استفاده کرد. الگوریتم Adam را میتوان به عنوان ترکیبی از RMSprop و گرادیان نزولی تصادفی با گشتاور (Momentum) در نظر گرفت. الگوریتم Adam دارای مزایای زیر است:

- این الگوریتم به آسانی پیادهسازی میشود.
- الگوریتم بهینه سازی Adam در برابر مقیاسدهی مجدد قطری گرادیانها تغییر نمیکند و به آنها وابستگی ندارد.
- Adam برای مسائلی بسیار مناسب است که به لحاظ داده یا پارامتر بزرگ محسوب میشوند.
- الگوریتم Adam برای اهداف ناایستا (غیر ثابت | غیرمانا) مناسب است.
- Adam الگوریتم خوبی برای مسائلی است که دارای گرادیانهایی بسیار نویزدار یا تُنُک (نامتراکم) هستند.

- «اَبَرپارامترها» (Hyper-Parameters) در الگوریتم بهینه سازی Adam دارای تفسیر بصری هستند و معمولاً به تنظیم پارامتر اندکی نیاز دارند.

الگوریتم گرادیان کاهشی تصادفی (SGD) یک الگوریتم کاهش شیب که در آن تعداد دادههای یک دسته برابر یک است. به بیان دیگر، این الگوریتم برای تخمین شیب در هر گام، تنها به یک نمونه داده که به صورت تصادفی از میان مجموعه داده انتخاب شده نیاز دارد. الگوریتم SGD دارای مزایای زیر است:

- این الگوریتم ساده و قابل فهم است.
- الگوریتم SGD برای مسائلی که دارای تابع هدف پیچیده و نامتقارن هستند، مناسب است.
- SGD میتواند از گیر کردن در نقاط زینی یا مینیممهای محلی جلوگیری کند، زیرا نویز موجود در گرادیانهای تصادفی میتواند به پیدا کردن نقاط بهتر کمک کند.
- SGD میتواند با مجموعه دادههای بزرگ و پیوسته کار کند، زیرا هر گام به یک نمونه داده نیاز دارد و نیازی به بارگذاری کل مجموعه داده نیست.

الگوریتم Adam و SGD را میتوان در برخی جنبهها مقایسه کرد:

- Adam معمولاً سریعتر از SGD به همگرایی میرسد، زیرا از تکنیکهایی مانند گشتاور و تنظیم نرخ یادگیری استفاده میکند که به بهبود سرعت و کیفیت بهینهسازی کمک میکنند.

ادام دیرتر وزن هارا اپدیت می کند و و نوسانات ان کم تر است.

- **Adam میتواند بهتر از SGD با گرادیانهای تُنُک یا نویزدار کنار بیاید، زیرا از تخمینهای گشتاور اول و دوم برای تنظیم نرخ یادگیری برای هر پارامتر به طور جداگانه استفاده میکند و نرخ یادگیری انعطاف پذیری دارد و با مسائل نویزی بهتر کنار می اید .ادام سریع تر از نقاط زینی رد می شود ولی پیچیده تر از SGD است .**
- Adam دارای بیشترین تعداد پارامترهای قابل تنظیم است، که میتواند هم مزیت و هم معایب داشته باشد. از یک طرف، این امکان را میدهد که الگوریتم را به شرایط مختلف سازگار کنیم، اما از طرف دیگر، ممکن است باعث افزایش پیچیدگی و زمان تنظیم شود.
- SGD دارای سادگی و شفافیت بیشتری نسبت به Adam است، که میتواند در برخی موارد مزیت داشته باشد. برای مثال، در مسائلی که دارای تابع هدف محدب هستند، SGD میتواند به راحتی به جواب بهینه برسد، در حالی که Adam ممکن است در نقاط غیر بهینه گیر کند. همچنین، SGD میتواند به روشن کردن مکانیزمهای موثر در یادگیری عمیق کمک کند.

ت)

سمت راست دسته ای کوچک و سمت چپ دسته ای است

به عنوان متقاطع بین GD و SGD در نظر گرفته می شود .در این رویکرد به جای تکرار در کل مجموعه داده یا یک مشاهده، مجموعه داده را به زیرمجموعه های کوچک (دسته ای) تقسیم می کنیم و گرادیان ها را برای هر دسته محاسبه می کنیم.

نزول گرادیان دسته ای در Batch Gradient Descent، تمام داده های آموزشی برای برداشتن یک مرحله در نظر گرفته می شود. میانگین گرادیان تمام مثال‌های آموزشی را می‌گیریم و سپس از آن گرادیان میانگین برای به‌روزرسانی پارامترهایمان استفاده می‌کنیم. بنابراین این فقط یک مرحله از شیب نزول در یک دوره است.

پس طبق تعاریف بالا در دسته ای یکنواخت تر و با نوسان کم تر حرکت می کنیم چون در هر بار بررسی میانگین کل دیتاها را حساب می کنیم و با یک نرخ مشخص کاهش می یابد زیان ما اما در دسته ای ممکن است دیتاهای انتخابی برای هر دسته متفاوت باشد و نوسانات مختلفی را برای دسته های مختلف شاهد باشم.

فرمول Mini Batch Gradient Descent که وزن‌های w را به‌روزرسانی می‌کند:

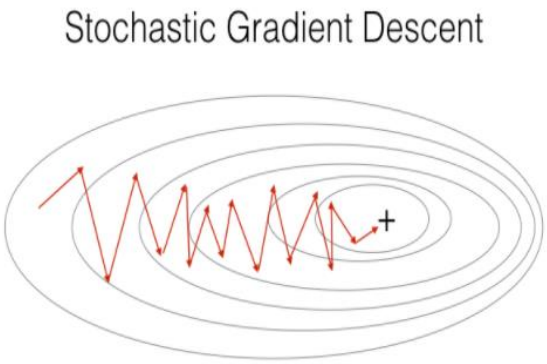
$$w_{i+1} = w_i - a \cdot \nabla_{w_i} J(x^{i:i+b}, y^{i:i+b}; w_i)$$

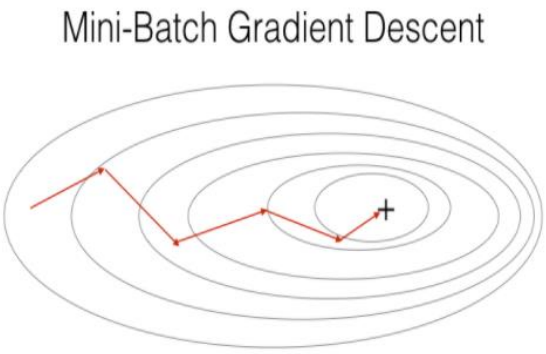
نزول گرادیان تصادفی (SGD) و شیب نزولی مینی دسته ای دو نوع از الگوریتم شیب نزولی هستند که برای بهینه سازی یک تابع با حرکت مکرر در جهت شیب دارترین نزول استفاده می شود .تفاوت اصلی بین SGD و نزول گرادیان مینی دسته ای، اندازه نمونه های داده ای است که برای محاسبه گرادیان در هر تکرار استفاده می شود SGD .تنها از یک نمونه داده (یا یک نقطه داده) برای محاسبه گرادیان و به روز رسانی پارامترها در هر تکرار استفاده می کند .این باعث می شود SGD سریع و کارآمد از نظر حافظه، اما پر سر و صدا و ناپایدار باشد . SGDمی تواند از مینیمم های محلی یا نقاط زینی فرار کند، اما ممکن است حول راه حل بهینه نیز نوسان کند و هرگز همگرا نشود .نزول گرادیان مینی دسته ای از یک زیر مجموعه کوچک از داده ها (یا یک دسته) برای محاسبه گرادیان و به روز رسانی پارامترها در هر تکرار استفاده می کند .این امر باعث می شود نزول گرادیان دسته ای پایدارتر و دقیق تر از SGD باشد، اما همچنین از نظر محاسباتی گران تر و حافظه فشرده تر است . نزول گرادیان دسته ای کوچک می تواند تعادل بین سرعت و دقت را متعادل کند و معمولاً در عمل این روش ترجیح داده می شود.

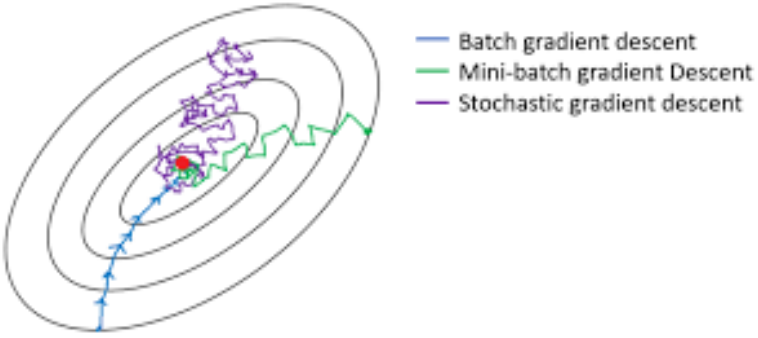
Batch Gradient Descent برای منیفولدهای خطای محدب یا نسبتا صاف عالی است. در این مورد، ما تا حدودی مستقیماً به سمت یک راه حل بهینه حرکت می کنیم. مینی دسته ای گرادیان نزول ما Batch Gradient Descent را دیده ایم. ما همچنین شاهد نزول گرادیان تصادفی هستیم. Batch Gradient Descent را می توان برای منحنی های صاف تر استفاده کرد. زمانی که مجموعه داده بزرگ باشد می توان از SGD استفاده کرد. Batch Gradient Descent مستقیماً به حداقل همگرا می شود. SGD برای مجموعه داده های بزرگتر سریعتر همگرا می شود. اما، از آنجایی که در SGD ما در هر زمان فقط از یک مثال استفاده می‌کنیم، نمی‌توانیم پیاده‌سازی برداری شده را روی آن پیاده‌سازی کنیم. این می تواند محاسبات را کند کند. برای مقابله با این مشکل، ترکیبی از Batch Gradient Descent و SGD استفاده می شود. نه ما از همه مجموعه داده به طور همزمان استفاده می کنیم و نه از مثال واحد در یک زمان استفاده می کنیم. ما از مجموعه‌ای از تعداد ثابت نمونه‌های آموزشی استفاده می‌کنیم که کمتر از مجموعه داده واقعی است و آن را یک دسته کوچک می‌نامیم. انجام این کار به ما کمک می کند تا به مزایای هر دو نوع قبلی که دیدیم دست پیدا کنیم. بنابراین، پس از ایجاد مینی بچ ها با اندازه ثابت، مراحل زیر را در یک دوره انجام می دهیم: یک مینی دسته انتخاب کنید آن را به شبکه عصبی تغذیه کنید میانگین گرادیان دسته کوچک را محاسبه کنید از گرادیان میانگینی که در مرحله ۳ محاسبه کردیم برای به روز رسانی وزن ها استفاده کنید مراحل ۱-۴ را برای مینی بچ هایی که ایجاد کردیم تکرار کنید دقیقاً مانند SGD، میانگین هزینه در طول دوره‌ها در نزول گرادیان دسته‌ای کوچک نوسان دارد، زیرا ما تعداد کمی از نمونه‌ها را در یک زمان میانگین می‌گیریم.

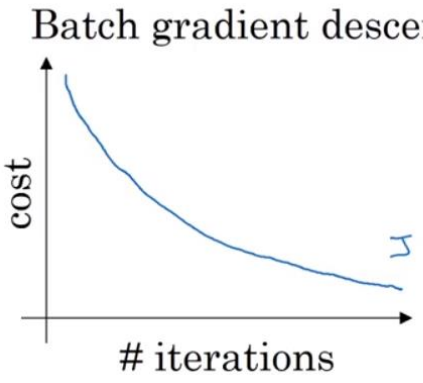
نزول شیب دسته ای و نزول شیب دسته ای کوچک دو نوع از الگوریتم شیب نزولی هستند که برای بهینه سازی یک تابع با حرکت مکرر در جهت شیب دارترین نزول استفاده می شود. تفاوت اصلی بین نزول گرادیان دسته ای و نزول گرادیان دسته ای کوچک، اندازه نمونه های داده ای است که برای محاسبه گرادیان در هر تکرار استفاده می شود. نزول گرادیان دسته ای از کل مجموعه داده برای محاسبه گرادیان و به روز رسانی پارامترها در هر تکرار استفاده می کند. این امر باعث می‌شود نزول شیب دسته‌ای دقیق‌تر و پایدارتر باشد، اما همچنین از نظر محاسباتی گران‌تر و حافظه فشرده‌تر است. نزول گرادیان دسته ای می تواند حداقل جهانی یک تابع محدب را پیدا کند، اما ممکن است در یک حداقل محلی یا یک نقطه زینی یک تابع غیر محدب نیز گیر کند. نزول گرادیان دسته ای کوچک از یک زیر مجموعه کوچک از مجموعه داده (یا یک دسته) برای محاسبه گرادیان و به روز رسانی پارامترها در هر تکرار استفاده می کند. این امر باعث می‌شود نزول گرادیان دسته‌ای پایدارتر و دقیق‌تر از نزول گرادیان تصادفی باشد، اما همچنین از لحاظ محاسباتی کارآمدتر و حافظه‌دارتر از نزول گرادیان دسته‌ای است. نزول گرادیان دسته ای کوچک می تواند تعادل بین سرعت و دقت را متعادل کند و معمولاً در عمل این روش ترجیح داده می شود.

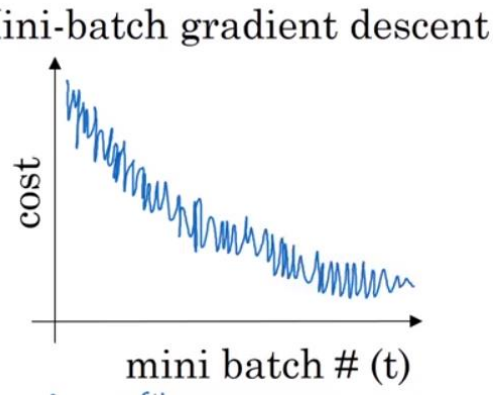
بنابراین، یک دسته برابر است با کل داده های آموزشی مورد استفاده در شیب نزول دسته ای برای به روز رسانی پارامترهای شبکه .از سوی دیگر، مینی دسته ای زیرمجموعه ای از داده های آموزشی است که در هر تکرار الگوریتم آموزشی در نزول گرادیان مینی دسته ای استفاده می شود.









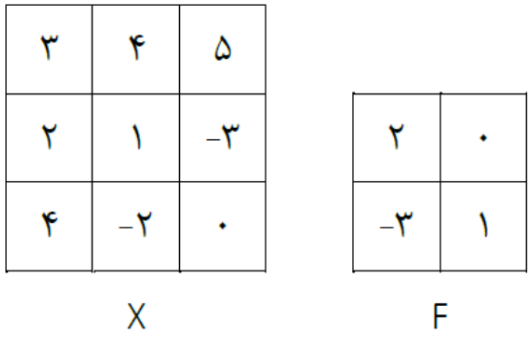


Mini Batch gradient descent: It which works faster than both batch gradient descent and stochastic gradient descent. Here *b* examples where *b<m* are processed per iteration. So even if the number of training examples is large, it is processed in batches of b training examples in one go. Thus, it works for larger training examples and that too with lesser number of iterations.

Batch Gradient Descent: It processes all the training examples for each iteration of gradient descent. If the number of training examples is large, then batch gradient descent is very expensive. So, in case of large training examples we prefer to use stochastic gradient descent or mini-batch gradient descent.

Variants of Gradient Descent Algorithm | Types of Gradient Descent (analyticsvidhya.com)

سوال ۲)



Forward

اعمال فیلتر F بر روی X داریم:

O11 = X11F11 + X12F12 + X21F21 + X22F22 =6+0-6+1=1

O12 = X12F11 + X13F12 + X22F21 + X23F22 = 8-3-3=2

O21 = X21F11 + X22F12 + X31F21 + X32F22 = 4-12-2= -10

O22 = X22F11 + X23F12 + X32F21 + X33F22 = 2+0+6+0=8

ماتریس خروجی مانند شکل زیر می شود

۱	۲
-۱۰	۸

اعمال global average pooling :

8 +1 +2 -10 =1

1 / 4 = 0.25

Backward

$$\frac{\partial L}{\partial O} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, \quad \frac{\partial L}{\partial F} = \frac{\partial L}{\partial O} * \frac{\partial O}{\partial F}, \quad \frac{\partial O}{\partial F} = X$$

∂ global average / ∂Ui =0.25

محاسبه قسمت بک وارد :

طبق توضیحات لینک داده شده جلو می رویم

ین معادله یک معادله مشتق جزئی است که در آن L یک تابع هزینه، F22 یک عنصر از یک فیلتر کانولوشنی، global یک مقدار خروجی از یک لایه کانولوشنی و Uj و Xij عناصر از یک لایه فعالسازی و یک لایه ورودی هستند. این معادله نشان میدهد که چگونه تغییر کوچکی در F22 میتواند تاثیری بر L داشته باشد. این تاثیر از طریق زنجیرهای شدن مشتقهای جزئی بر حسب global، Uj و F22 بدست میآید. این معادله میتواند برای بهروزرسانی F22 با استفاده از روش گرادیان کاهشی مورد استفاده قرار گیرد.

$\partial L / \partial F11 = \partial L / \partial global * (\sum_j \partial\ global / \partial\ Uj * \partial\ Uj / \partial\ F11) = 1 * (0.25 *X11 +0.25 *X12 + 0.25 *X21 + 0.25*X22)= 0.25 * 10 = 2.5$

$\partial L / \partial F12 = \partial L / \partial global * (\sum_j \partial\ global / \partial\ Uj * \partial\ Uj / \partial\ F12) = 1 * (0.25 *X12 +0.25 *X13 + 0.25 *X22 + 0.25*X23)= 0.25 * 7 = 1.75$

$\partial L / \partial F22 = \partial L / \partial global * (\sum_j \partial\ global / \partial\ Uj * \partial\ Uj / \partial\ F22) = 1 * (0.25 *X22 +0.25 *X32 + 0.25 *X23 + 0.25*X33)= -1$

$\partial L / \partial F21 = \partial L / \partial global * (\sum_j \partial\ global / \partial\ Uj * \partial\ Uj / \partial\ F21) = 1 * (0.25 *X22 +0.25 *X32 + 0.25 *X31 + 0.25*X21)= 0.25(5)=1.25$

2.5	1.75
1.25	-1

یک تابع L .داشته باشد L میتواند تاثیری بر X11 ین معادله نشان میدهد که چگونه تغییر کوچکی در یک X11 یک عنصر از یک لایه فعالسازی و Ujیک مقدار خروجی از یک لایه کانولوشنی، globalهزینه، و Uj، global عنصر از یک لایه ورودی است. این تاثیر از طریق زنجیرهای شدن مشتقهای جزئی بر حسب با استفاده از روش گرادیان کاهشی مورد X11 بدست میآید. این معادله میتواند برای بهروزرسانی X11 استفاده قرار گیرد

$\partial L / \partial X11 = \partial L / \partial global * (\sum_j \partial\ global / \partial\ Uj * \partial\ Uj / \partial\ X11) = 1 * (0.25 *F11 +0.25 *0 + 0.25 *0 + 0.25*0)= 0.5$

۵

برای

X12 , X13 , X21 , X22 , X23 , X31 , X32 , X33

این کار را انجام می دهیم تا ماتریس ۳ ۳ بدست اید .

۰	۰,۵	۰,۵
۰,۲۵	۰	۱,۲۵
۰,۲۵	-0.5	-0.75

$\partial L / \partial X22 = \partial L / \partial global * (\sum_j \partial\ global / \partial\ Uj * \partial\ Uj / \partial\ X22) = 1 * (0.25 *F22 +0.25 *F21 + 0.25 *F11+ 0.25*F12)= 0$

$\partial L / \partial X33 = \partial L / \partial global * (\sum_j \partial\ global / \partial\ Uj * \partial\ Uj / \partial\ X33) = 1 * (0.25 *F22 +0.25 *0 + 0.25 *0+ 0.25*0)= 0.25$

$\partial L / \partial X12 = \partial L / \partial global * (\sum_j \partial\ global / \partial\ Uj * \partial\ Uj / \partial\ X12) = 1 * (0.25 *F12 +0.25 *F11 + 0.25 *0+ 0.25*0)= 0.5$

$\partial L / \partial X13 = \partial L / \partial global * (\sum_j \partial\ global / \partial\ Uj * \partial\ Uj / \partial\ X13) = 1 * (0.25 *F12 +0.25 *0 + 0.25 *0+ 0.25*0)= 0$

$\partial L / \partial X21 = \partial L / \partial global * (\sum_j \partial\ global / \partial\ Uj * \partial\ Uj / \partial\ X21) = 1 * (0.25 *0 +0.25 *F21 + 0.25 *0+ 0.25*F11)= 1.25$

$\partial L / \partial X23 = \partial L / \partial global * (\sum_j \partial\ global / \partial\ Uj * \partial\ Uj / \partial\ X23) = 1 * (0.25 *F22 +0.25 *F12 + 0.25 *0+ 0.25*0)= 0.25$

$\partial L / \partial X31 = \partial L / \partial global * (\sum_j \partial\ global / \partial\ Uj * \partial\ Uj / \partial\ X31) = 1 * (0.25 *F21+0.25 *0 + 0.25 *0+ 0.25*0)= -0.75$

$\partial L / \partial X32 = \partial L / \partial global * (\sum_j \partial\ global / \partial\ Uj * \partial\ Uj / \partial\ X32) = 1 * (0.25 *F22 +0.25 *F21 + 0.25 *0+ 0.25*0)= -0.5$

سوال ۳)

الف)

$S=1 , p =0$

طبق پاسخنامه کوییز عمل می کنیم .

.۱

For convolution:

$$\text{Output size} = \frac{\text{Input size} - \text{Filter size} + 2 \times \text{Padding}}{\text{Stride}} + 1$$

For pooling:

$$\text{Output size} = \frac{\text{Input size} - \text{Pooling size}}{\text{Stride}} + 1$$

Convolutional layer parameters:

$$\text{Parameters} = (\text{Filter size} \times \text{Filter size} \times \text{Input depth} + 1) \times \text{Number of filters}$$

Fc layer parameters:

If you have a Flatten layer followed by an FC layer with N neurons, and the flattened input has size M, then the number of parameters (P) in the FC layer is given by:

$$P = M \times N + N$$

layer	parametr
Conv1D	Filters * (kernel size * input channel size +1)
MaxPool1D	0
Flatten	0
Dence	(Input size +1) * units

500-3 +1 =498

(498-2) /2 +1 = 249

249 -5 +1 = 245

(245-2)/2 +1 =122

(122-5)+1 =118

(118-2)/2 +1 =59

Layer	Activation Volume Dimensions	Number of parameters
Input	500,7	0
Conv1D	498,16	16*(3*7+1) = 352
MaxPool1D	249,16	0
Conv1D	245,32	32*(5*16+1) = 2592
MaxPool1D	122,32	0
Conv1D	118,64	64*(5*32+1) = 10304
MaxPool1D	59,64	0
Flatten	3776	0
FC - 128	128	128*(3776 + 1) = 483456
FC - 5	5	5*(128 + 1) = 645
total		497349

ب)

لایه همگشتی دوبعدی (D2Conv) و لایه همگشتی سه بعدی (D3Conv) دو نوع لایه همگشتی هستند که در شبکه های عصبی همگشتی استفاده میشوند. تفاوت اصلی بین این دو لایه در ابعاد ورودی و خروجی آنها است. لایه همگشتی دوبعدی ورودی هایی را دریافت میکند که دارای دو بعد مکانی و یک بعد کانال هستند، مانند تصاویر خاکستری یا رنگی، و خروجی هایی را تولید میکند که همچنان دارای دو بعد مکانی و یک بعد کانال هستند، مانند نقشه های ویژگی. لایه همگشتی دوبعدی از یک فیلتر همگشتی دوبعدی استفاده میکند که به صورت محلی بر روی ورودی عمل میکند و یک عدد اسکالر را به عنوان خروجی می دهد.

لایه همگشتی سه بعدی ورودیهایی را دریافت میکند که دارای سه بعد مکانی و یک بعد کانال هستند، مانند ویدیوها یا تصاویر سهبعدی، و خروجیهایی را تولید میکند که همچنان دارای سه بعد مکانی و یک بعد کانال هستند، مانند نقشههای ویژگی سهبعدی. لایه همگشتی سهبعدی از یک فیلتر همگشتی سهبعدی استفاده میکند که به صورت محلی بر روی ورودی عمل میکند و یک عدد اسکالر را به عنوان خروجی میدهد.

لایه همگشتی سه بعدی میتواند برای کاربردهایی که نیاز به استخراج ویژگیهای مکانی و زمانی از دادههای سه بعدی دارند، مفید باشد. برای مثال، لایه همگشتی سه بعدی میتواند برای تشخیص اشیا در ویدیوها، تحلیل حرکت انسان، بازشناسی چهره سه بعدی، تشخیص بیماریهای ریه از تصاویر CT و غیره استفاده شود.

شبکه عصبی کانولوشن سه بعدی یک مدل یادگیری عمیق است که در کاربردهای مختلف مانند بینایی کامپیوتر یا تصویربرداری پزشکی استفاده می شود.

در این موارد، ما می خواهیم هوش مصنوعی (یادگیری عمیق) یاد بگیرد که چگونه به ورودی ها واکنش نشان دهد نه اینکه هوش مصنوعی را بر اساس یک الگوی از پیش تعیین شده برنامه ریزی کند. نتیجه این فرآیند یادگیری یک مدل پیش بینی است. مدل‌های پیش‌بینی‌شده از چارچوب شبکه عصبی کانولوشنال سه‌بعدی برای پردازش و تجزیه و تحلیل داده‌ها با ابعاد زمانی، مانند ویدیوها، طراحی شده‌اند. شبکه های عصبی کانولوشنال سه بعدی را می توان برای پردازش داده های ابر نقطه سه بعدی از حسگرهای LiDAR برای تشخیص اشیا و وظایف بخش بندی معنایی در رباتیک و وسایل نقلیه خودمختار استفاده کرد.

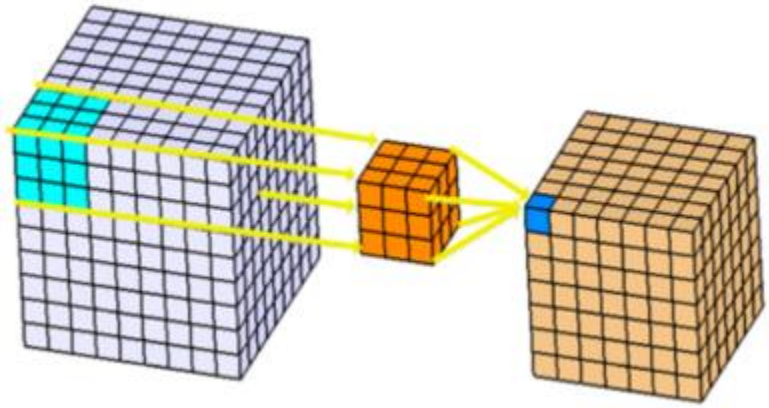
CNN سه بعدی می تواند روابط فضایی را در داده ها یاد بگیرد و ویژگی هایی را استخراج کند که می تواند برای کارهایی مانند طبقه بندی یا تقسیم بندی استفاده شود. هدف این است که یک برچسب معنایی مانند "جاده"، "ساختمان"، "خودرو" و غیره به هر پیکسل در تصویر اختصاص دهیم تا درک دقیقی از اطلاعات مکانی در صحنه ارائه شود. یک نوآوری اخیر، استفاده از یک شبکه عصبی کانولوشنال سه بعدی برای مدل‌های یادگیری عمیق است تا نشان دهد چگونه نرم‌افزار شبیه‌سازی می‌تواند پیش‌بینی‌های مهندسی دقیق را با طرح‌های CAD، همچنین با عدم قطعیت پیش‌بینی، مرتبط کند.

2D Convolutions

On image datasets, mostly 2D Convolutional filters are used in CNN architectures. The main idea of 2D convolutions is that the the convolutional filter moves in 2-directions (x,y) to calculate low dimentional features from the image data. The output shape is also a 2 dimentional matrix.

3D Convolutions

3D convolutions applies a 3 dimentional filter to the dataset and the filter moves 3-direction (x, y, z) to calcuate the low level feature representations. Their output shape is a 3 dimentional volume space such as cube or cuboid. They are helpful in event detection in videos, 3D medical images etc. They are not limited to 3d space but can also be applied to 2d space inputs such as images.



Lets implement the 3D CNN on 3D Mnist dataset. First, lets import the key libraries.

The only difference is the dimensionality of the input space. The input for a convolutional layer has the following shape:

input_shape = (batch_size,input_dims,channels)

- Input shape for **conv1D**: (batch_size,W,channels)

  Example: 1 second stereo voice signal sampled at 44100 Hz, shape: (batch_size,44100,2)

- Input shape for **conv2D**: (batch_size,(H,W),channels)

  Example: 32x32 RGB image, shape: (batch_size,32,32,3)

- Input shape for **conv3D**: (batch_size,(H,w,D),channels)

  Example (more tricky): 1 second video of 32x32 RGB images at 24 fps, shape: (batch_size,32,32,3,24)

Logically, if we talk about video-based tasks, Conv2d may not observe temporal context and each frame will be processed independently (except they might interact during Batchnorm). Conv3d would make the features from each frame interact and may learn temporal contexts. Empirically, Conv3d is shown to perform better.

2D

 The convolutional kernel moves in 2-direction (x,y) to calculate the convolutional output.

 The output shape of the output is a 2D Matrix.

 Use cases: Image Classification, Generating New Images, Image Inpainting, Image Colorization, etc.

3D

The convolutional kernel moves in 3-direction (x,y,z) to calculate the convolutional output. Use Case: Conv3D is mostly used with 3D image data such as Magnetic Resonance Imaging (MRI) or Computerized Tomography (CT) Scan.

https://towardsdatascience.com/conv1d-and-conv2d-did-you-realize-that-conv1d-is-a-subclass-of-conv2d-8819675bec78

سوال ۴)

قسمت اول

قسمت اول: در این قسمت کتابخانههای مورد نیاز را وارد میکنیم. تنسورفلو یک کتابخانه محبوب برای یادگیری عمیق است که امکان ساخت و آموزش مدلهای پیچیده را فراهم میکند. کراس یک واسط بالا سطح برای تنسورفلو است که کار با لایهها، مدلها و توابع خسارت و بهینهسازی را آسانتر میکند. متپلاتلیب یک کتابخانه برای رسم نمودارها و تصاویر است. gdown یک کتابخانه برای دانلود فایلها از گوگل درایو است.

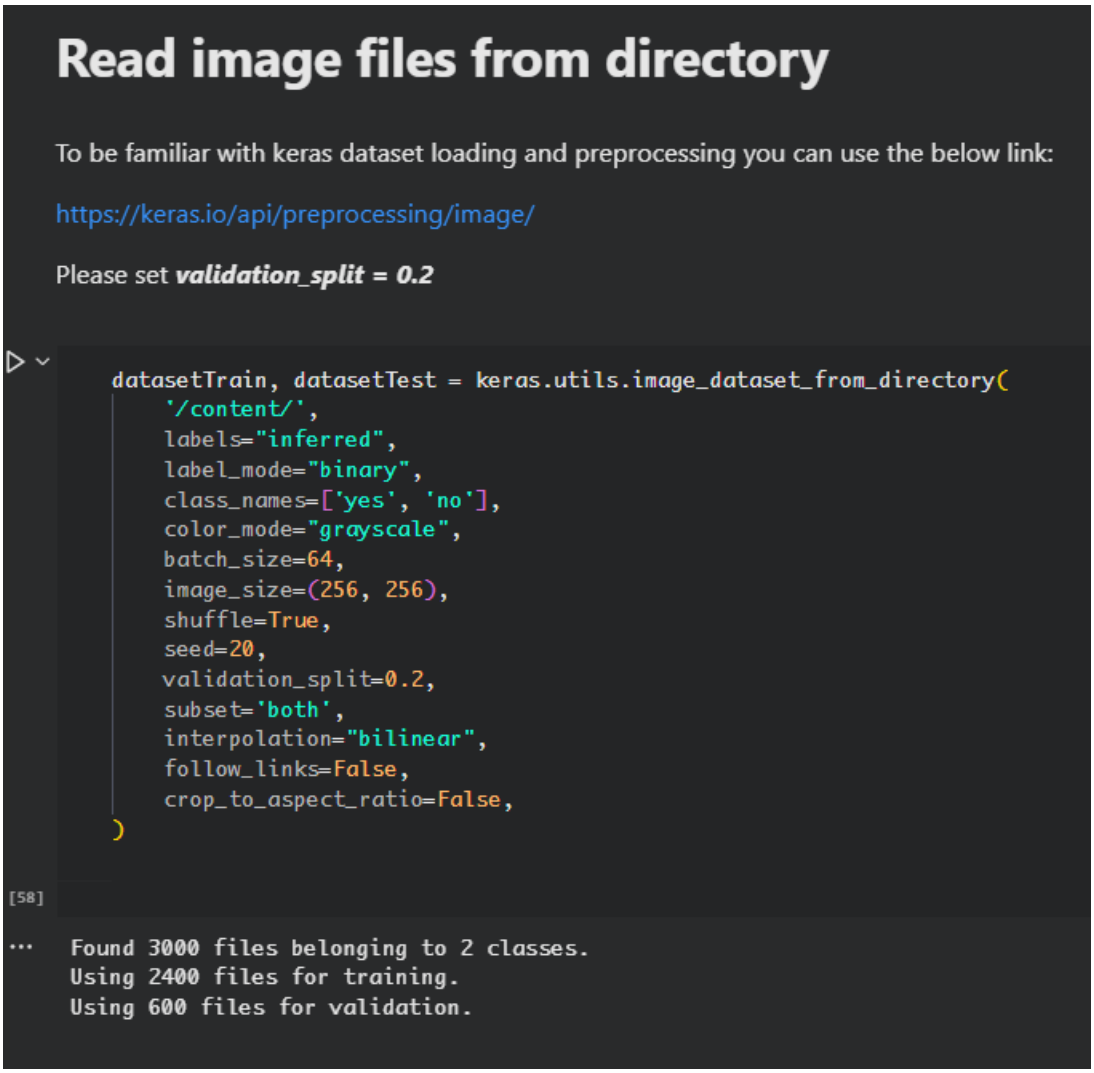
Import libraries

```
import tensorflow as tf
from tensorflow import keras
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import Dense, Conv2D, MaxPool2D, Activation, Input, Flatten, Rescaling
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping

import gdown
```

قسمت دوم: در این قسمت مجموعه داده را از گوگل درایو دانلود و از حالت فشرده خارج میکنیم. مجموعه داده شامل تصاویری از افرادی است که در حال خم شدن هستند یا نیستند. تصاویر در دو پوشه yes و no قرار دارند. همچنین پوشه sample_data را که به صورت پیشفرض در کولب وجود دارد، حذف میکنیم.

یو ار ال گذاشته شده را ومسیر فایل output را در قسمت دانلود دیتاست گذاشتم

این کد یک کد پایتون است که با استفاده از کتابخانه کراس، یک دیتاست از تصاویر را بارگذاری میکند. این دیتاست شامل تصاویری از دو دسته yes و no است که نشان میدهند که آیا یک تصویر شامل یک چیز خاص است یا خیر. این کد از تابع image_dataset_from_directory برای ساختن دیتاستهای آموزشی و تست از پوشههای موجود در مسیر /content/ استفاده میکند. این تابع چندین پارامتر را دریافت میکند که توضیح آنها به شرح زیر است:



directory: این پارامتر نشان میدهد که مجموعه داده در کدام پوشه قرار دارد. در این مثال، مسیر پوشه '/content/' است.

- labels: این پارامتر نشان میدهد که چگونه برچسبهای تصاویر را تعیین کنیم. در این مثال، مقدار 'inferred' به این معنی است که برچسبها بر اساس نام پوشههای زیرمجموعه استخراج میشوند.

- label_mode: این پارامتر نشان میدهد که چگونه برچسبها را ذخیره کنیم. در این مثال، مقدار 'binary' به این معنی است که برچسبها به صورت یک عدد صحیح ۰ یا ۱ ذخیره میشوند.

- class_names: این پارامتر نشان میدهد که چه نامهایی را برای کلاسهای مختلف استفاده کنیم. در این مثال، دو کلاس 'yes' و 'no' وجود دارند.

• color_mode: این پارامتر نشان میدهد که چه حالت رنگی را برای تصاویر استفاده کنیم. در این مثال، مقدار 'grayscale' به این معنی است که تصاویر به صورت سیاه و سفید بارگذاری میشوند.

• batch_size: این پارامتر نشان میدهد که چه تعداد تصویر را در هر دسته (batch) بارگذاری کنیم. در این مثال، مقدار ۶۴ به این معنی است که هر ۶۴ تصویر یک دسته را تشکیل میدهند.

• image_size: این پارامتر نشان میدهد که چه اندازهای را برای تصاویر در نظر بگیریم. در این مثال، مقدار (۲۵۶, ۲۵۶) به این معنی است که تصاویر به ابعاد ۲۵۶ در ۲۵۶ پیکسل تغییر اندازه مییابند.

• shuffle: این پارامتر نشان میدهد که آیا تصاویر را به صورت تصادفی مرتب کنیم یا خیر. در این مثال، مقدار True به این معنی است که تصاویر به صورت تصادفی مرتب میشوند.

• seed: این پارامتر نشان میدهد که چه عددی را به عنوان دانه (seed) برای تولید تصادفی استفاده کنیم. در این مثال، مقدار ۲۰ به این معنی است که تولید تصادفی با دانه ۲۰ انجام میشود.

• validation_split: این پارامتر نشان میدهد که چه نسبتی از تصاویر را برای مجموعه داده آزمون (validation) جدا کنیم. در این مثال، مقدار ۰/۲ به این معنی است که ۲۰ درصد از تصاویر برای مجموعه داده آزمون استفاده میشوند.

• subset: این پارامتر نشان میدهد که کدام بخش از مجموعه داده را برگردانیم. در این مثال، مقدار 'both' به این معنی است که هم مجموعه داده آموزشی (training) و هم مجموعه داده آزمون را برگردانیم.

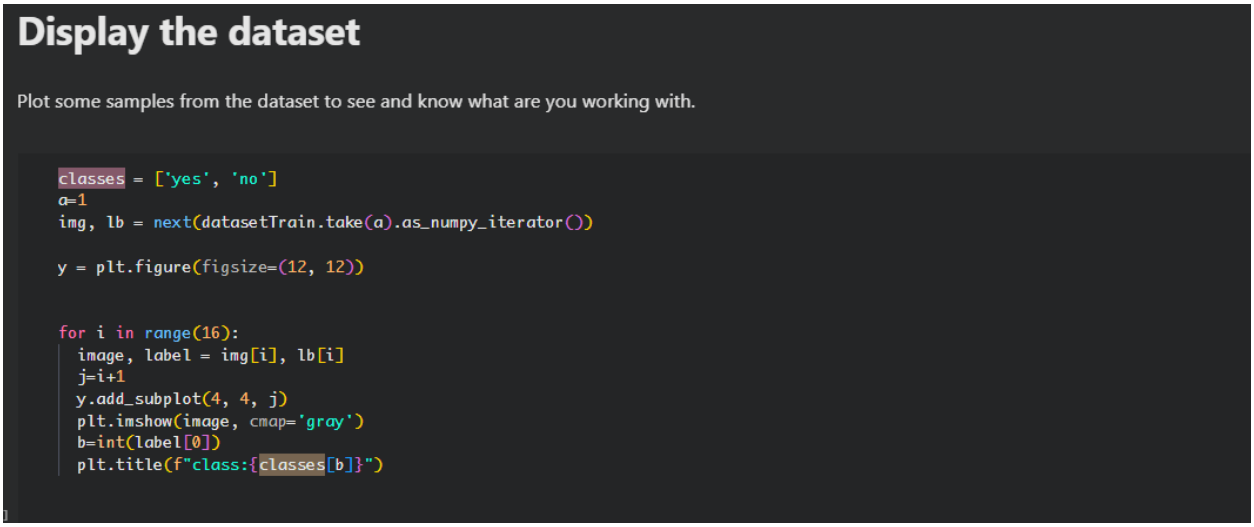
• interpolation: این پارامتر نشان میدهد که چه روشی را برای تغییر اندازه تصاویر استفاده کنیم. در این مثال، مقدار 'bilinear' به این معنی است که از روش خطی دوگانه (bilinear interpolation) استفاده میشود که یک روش متداول برای تغییر اندازه تصاویر است.

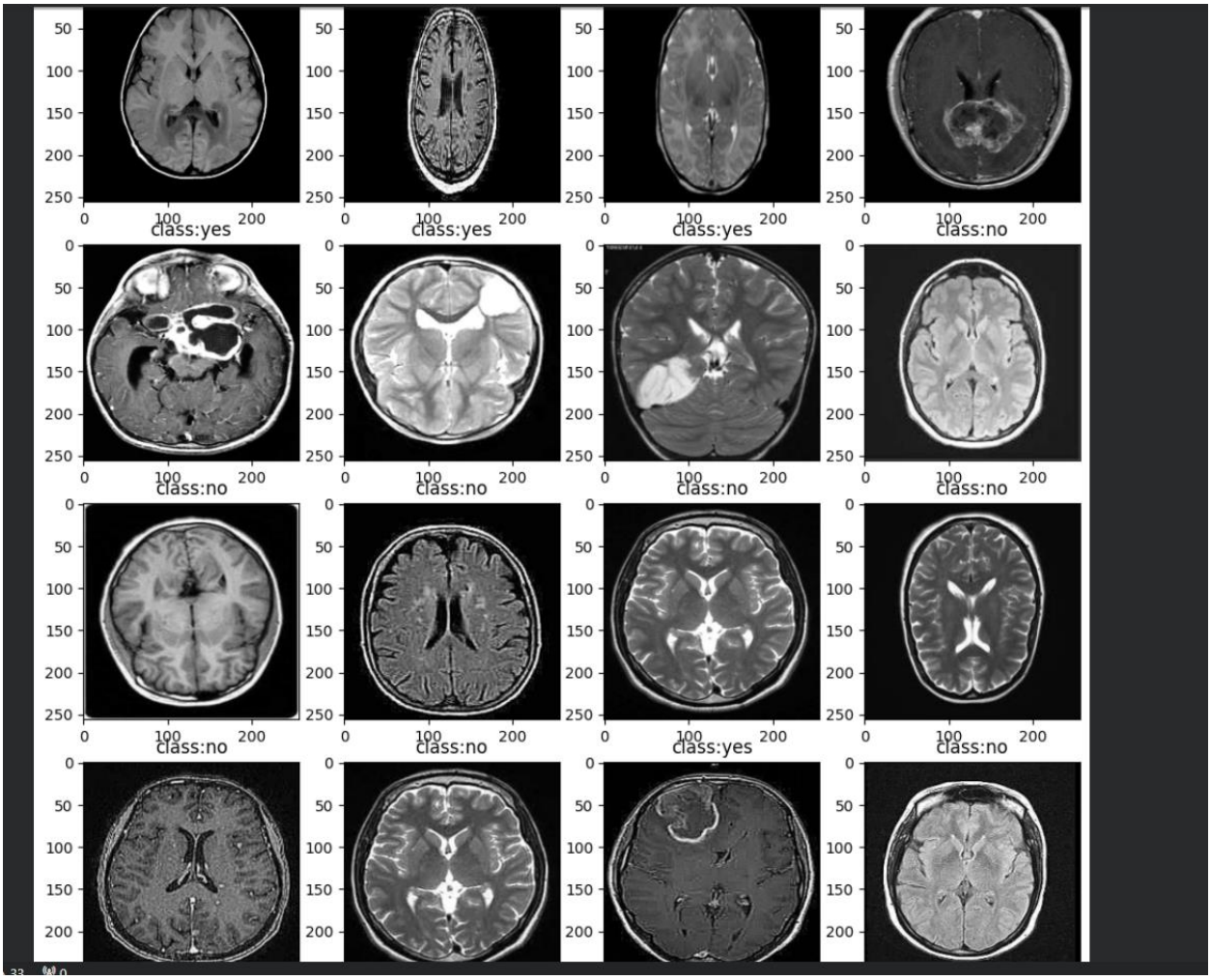
• follow_links: این پارامتر نشان میدهد که آیا تصاویر را از پیوندهای نمادین (symbolic links) دنبال کنیم یا خیر. در این مثال، مقدار False به این معنی است که تصاویر را از پیوندهای نمادین دنبال نمیکنیم.

• crop_to_aspect_ratio: این پارامتر نشان میدهد که آیا تصاویر را بر اساس نسبت ابعاد (aspect ratio) برش دهیم یا خیر. در این مثال، مقدار False به این معنی است که تصاویر را بر اساس نسبت ابعاد برش نمیدهیم.

این کد دو شیء از نوع tf.data.Dataset را برمیگرداند که مجموعه داده آموزشی و مجموعه داده آزمون را نشان میدهند. این شیءها را میتوانیم برای آموزش و ارزیابی مدلهای شبکه عصبی استفاده کنیم.

۱۶ تصویر را به همراه کلاس ان ع=ها به صورت سیاه سفید نمایش می دهیم در هر بچ





قسمت دوم

ابتدا پیاده سازی سکونشال

روش Sequential API و روش Functional API. این دو روش دو روش متفاوت برای تعریف معماری مدل هستند. روش Sequential API یک روش ساده و خطی است که لایهها را به ترتیب پشت سر هم قرار میدهد. روش Functional API یک روش پیچیدهتر و انعطافپذیرتر است که امکان ایجاد مدلهای با ساختارهای غیرخطی را میدهد.

ین کد یک مدل شبکه عصبی پیچشی (CNN) را با استفاده از کتابخانه TensorFlow ایجاد میکند. مدل CNN برای دستهبندی تصاویر دودویی (بیناری) طراحی شده است. مدل CNN شامل لایههای زیر است:

• لایه ورودی: این لایه شکل تصاویر ورودی را مشخص میکند. در این مثال، تصاویر ورودی دارای ابعاد ۲۵۶ در ۲۵۶ پیکسل و یک کانال رنگی هستند.

• لایههای پیچشی: این لایهها با استفاده از فیلترهایی با اندازه ۳ در ۳ پیکسل، ویژگیهای مهم تصاویر را استخراج میکنند. تعداد فیلترها در هر لایه پیچشی به ترتیب ۱۶، ۳۲ و ۶۴ است. همچنین از تابع فعالسازی relu برای افزایش خطیسازی استفاده شده است.

• لایههای حداکثر تجمیع (MaxPool): این لایهها با استفاده از پنجرههایی با اندازه ۲ در ۲ پیکسل، اندازه تصاویر را کاهش میدهند و از ویژگیهای مهم آنها حفظ میکنند. این لایهها بعد از هر لایه پیچشی قرار دارند.

• لایه صافی (Flatten): این لایه تصاویر را از حالت دوبعدی به حالت یکبعدی تبدیل میکند تا بتوانند به لایههای تماما متصل (Dense) متصل شوند.

• لایههای تماما متصل: این لایهها با استفاده از یادگیری عمیق، روابط بین ویژگیهای تصاویر را یاد میگیرند. تعداد نورونها در این لایهها به ترتیب ۶۴ و ۱ است. از تابع فعالسازی relu برای لایه اول و از تابع فعالسازی sigmoid برای لایه آخر استفاده شده است. تابع فعالسازی sigmoid مقدار خروجی را بین ۰ و ۱ قرار میدهد که میتواند به عنوان احتمال تعلق به یکی از دو کلاس تفسیر شود. , و دسته بندی باینری داریم

مدل CNN را میتوانید در شکل زیر مشاهده کنید.

برای آموزش مدل CNN، از تابع خطای binary_crossentropy برای محاسبه اختلاف بین پیشبینیها و برچسبهای واقعی استفاده شده است. همچنین از بهینهساز adam برای بهروزرسانی وزنهای مدل با استفاده از الگوریتم گرادیان نزولی استفاده شده است. معیار ارزیابی مدل CNN دقت (accuracy) است که نشان میدهد چه تعداد از تصاویر را مدل به درستی دستهبندی کرده است.

پس از آموزش مدل CNN، میتوانیم دقت و خطای مدل را روی دادههای آموزشی و آزمون محاسبه کنیم. برای این کار، از تابع evaluate مدل استفاده میکنیم. این تابع خروجیهای مدل را با برچسبهای واقعی مقایسه میکند و مقدار خطا و دقت را برمیگرداند. ما این مقادیر را برای دادههای آموزشی و آزمون چاپ میکنیم.

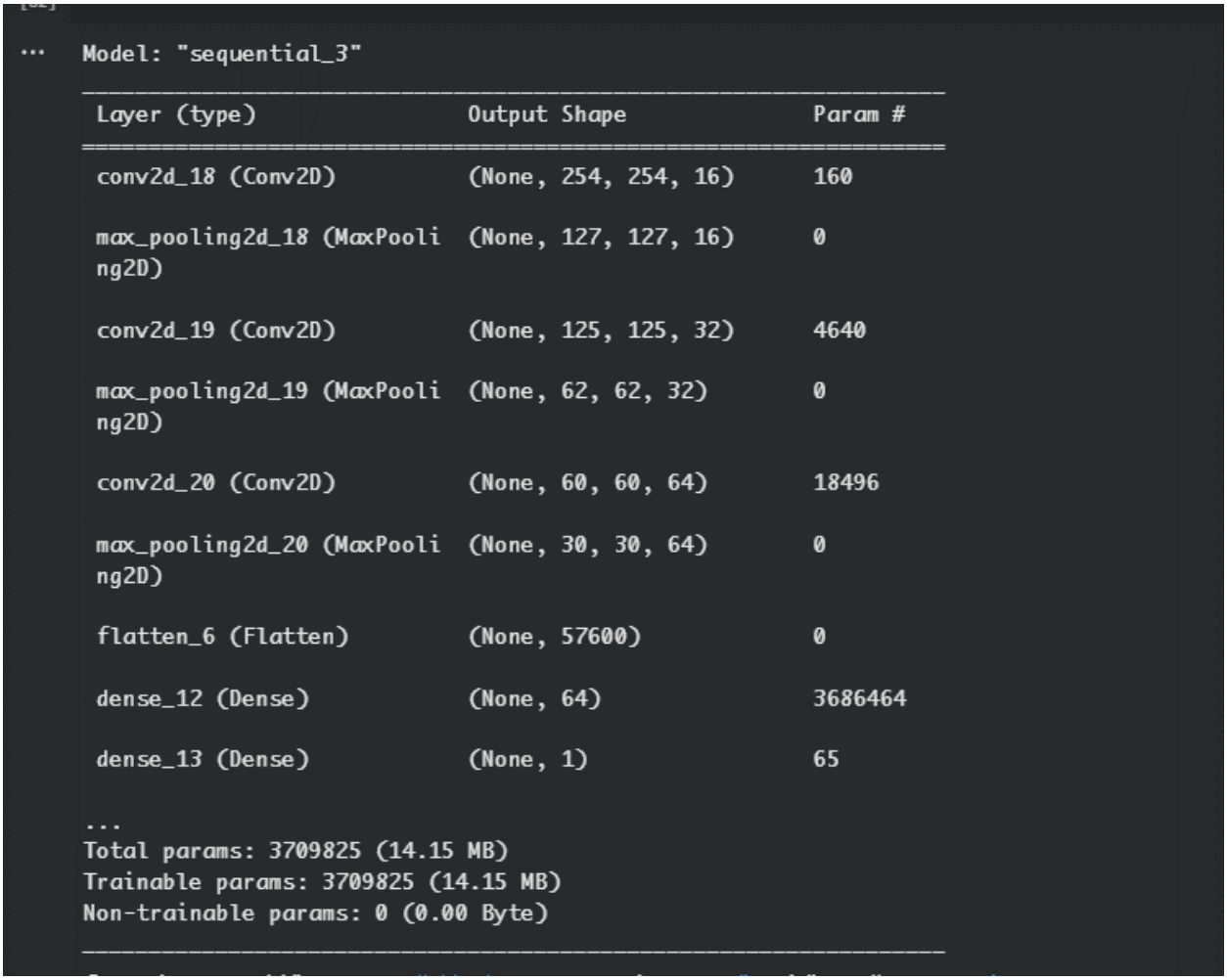
1.Sequential API

Build a model

```
model_seq = Sequential([
    Input(shape=(256, 256, 1)),
    Conv2D(16, (3, 3), activation='relu'),
    MaxPool2D((2, 2)),
    Conv2D(32, (3, 3), activation='relu'),
    MaxPool2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPool2D((2, 2)),
    Flatten(),
    Dense(64, activation='relu'),
    Dense(1, activation='sigmoid')
])
model_seq.summary()
tf.keras.utils.plot_model(model_seq, show_shapes=True)
```

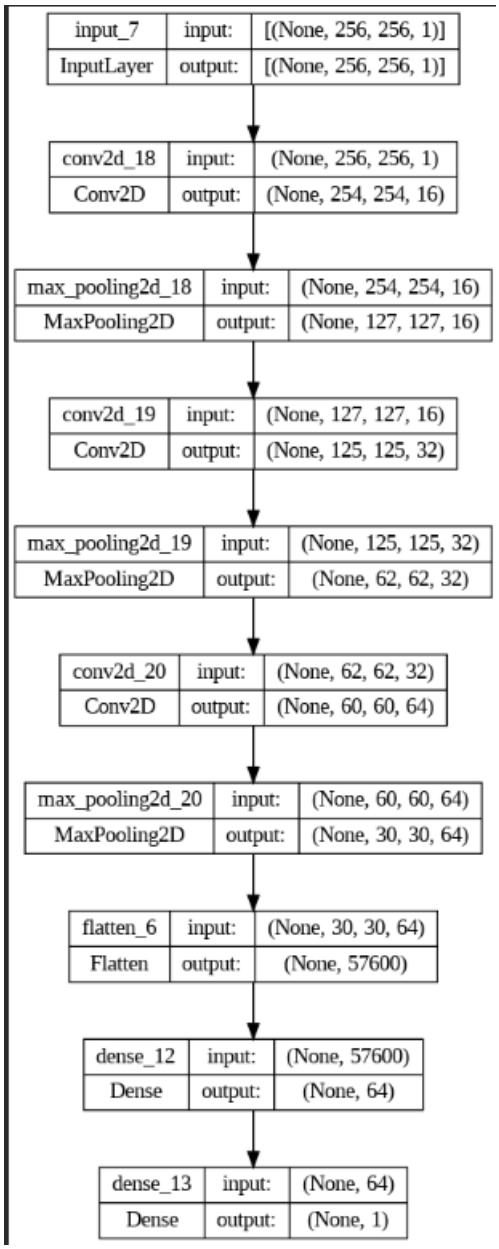
از توابع فعال ساز رلو و سیگموید استفاده شده و دقیقا همین لایه و با ساختار فانکشنال در قسمت فانکشنال پیاده سازی شده

و در اخر هم مشاهده می شود که دقت و لاس بسیار شبیه بهم با اختلاف بسیار کمی دارند که مقدار لاس و دقت هر دو در کادر های زرد اورده شده برای مقایسه دلیل این اختلاف کم هم برای این است که ساختار کلی شبکه مشترک است و فقط در نحوه پیاده سازی تفاوت دارند نمودار ها هم شبیه بهم است



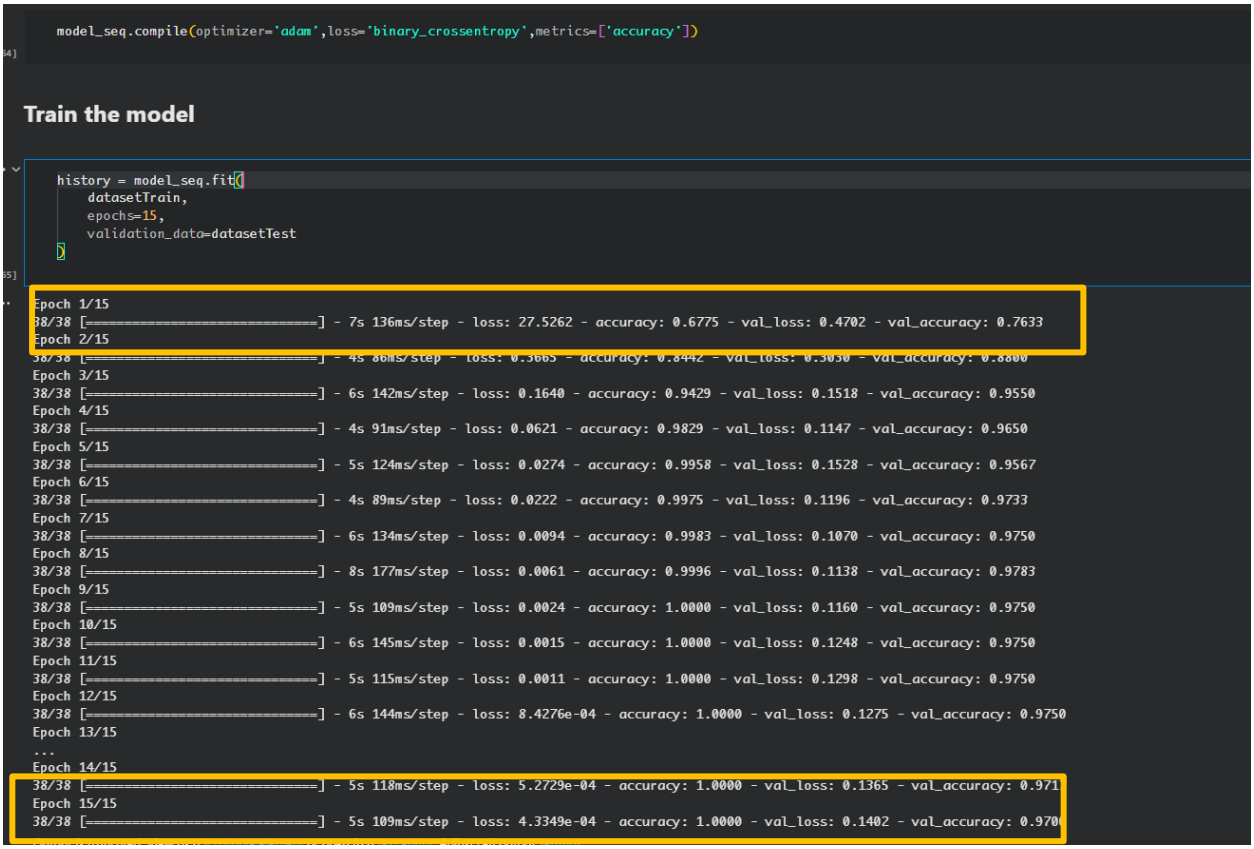
```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_18 (Conv2D)          (None, 254, 254, 16)      160

 max_pooling2d_18 (MaxPooli  (None, 127, 127, 16)      0
 ng2D)

 conv2d_19 (Conv2D)          (None, 125, 125, 32)      4640

 max_pooling2d_19 (MaxPooli  (None, 62, 62, 32)        0
 ng2D)

 conv2d_20 (Conv2D)          (None, 60, 60, 64)        18496

 max_pooling2d_20 (MaxPooli  (None, 30, 30, 64)        0
 ng2D)

 flatten_6 (Flatten)         (None, 57600)             0

 dense_12 (Dense)            (None, 64)                3686464

 dense_13 (Dense)            (None, 1)                 65

...
Total params: 3709825 (14.15 MB)
Trainable params: 3709825 (14.15 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

مقدار هر لایه به همراه پارامتر و دیمانسیون ان می بینیم که از همگشتی و پولینگ استفاده شده در ۹ لایه بدون در نظر گرفتن پولینگ ها ۶ لایه که می توانستیم لایه ها را بیش تر کنیم اما با همین مقدار هم به دقت خوبی میرسیم

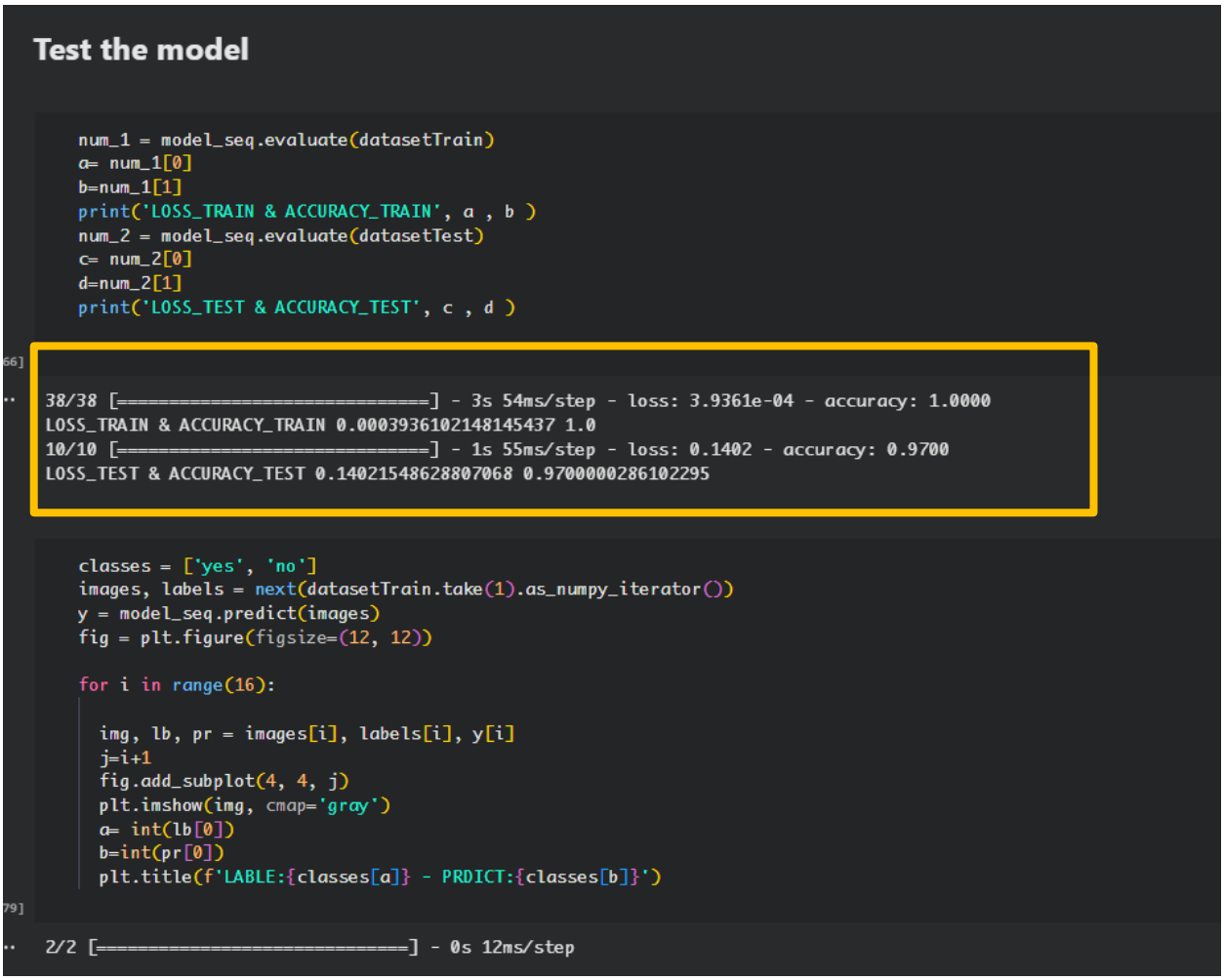

input_7 | input: | [(None, 256, 256, 1)]
InputLayer | output: | [(None, 256, 256, 1)]
conv2d_18 | input: | (None, 256, 256, 1)
Conv2D | output: | (None, 254, 254, 16)
max_pooling2d_18 | input: | (None, 254, 254, 16)
MaxPooling2D | output: | (None, 127, 127, 16)
conv2d_19 | input: | (None, 127, 127, 16)
Conv2D | output: | (None, 125, 125, 32)
max_pooling2d_19 | input: | (None, 125, 125, 32)
MaxPooling2D | output: | (None, 62, 62, 32)
conv2d_20 | input: | (None, 62, 62, 32)
Conv2D | output: | (None, 60, 60, 64)
max_pooling2d_20 | input: | (None, 60, 60, 64)
MaxPooling2D | output: | (None, 30, 30, 64)
flatten_6 | input: | (None, 30, 30, 64)
Flatten | output: | (None, 57600)
dense_12 | input: | (None, 57600)
Dense | output: | (None, 64)
dense_13 | input: | (None, 64)
Dense | output: | (None, 1)

مدل CNN را با استفاده از دادههای آموزشی (datasetTrain) و با تعداد ۱۵ دوره (epoch) آموزش میدهیم. همچنین از دادههای آزمون (datasetTest) برای ارزیابی عملکرد مدل در هر دوره استفاده میکنیم.

مقدار لاس کم شده دقت افزایش داشته




```
inputs_model = tf.keras.Input(shape=(256, 256, 1))
x = Conv2D(16, (3, 3), activation='relu')(inputs_model)
x = MaxPool2D((2, 2))(x)
x = Conv2D(32, (3, 3), activation='relu')(x)
x = MaxPool2D((2, 2))(x)
x = Conv2D(64, (3, 3), activation='relu')(x)
x = MaxPool2D((2, 2))(x)
x = Flatten()(x)
x = Dense(64, activation='relu')(x)
outputs_model = Dense(1, activation='sigmoid')(x)

model_func = tf.keras.Model(inputs=inputs_model, outputs=outputs_model)
model_func.summary()
```



در این قسمت هم تفاوت و میزان مقدار خطا و دقت داده های اموزش تست را میبینیم و در ادامه ۱۶ تصویر به همراه لیبل و کلاس پیش بینی شده برای هر کدام اوردیم

برچسب واقعی هر تصویر را در عنوان نمایش میدهد. این کد میتواند به ما نشان دهد که مدل CNN چقدر قادر است تصاویر را به درستی دستهبندی کند و کجا اشتباه میکند.

asses = ['yes', 'no'] این کد یک لیست از دو عنصر میسازد که میتواند به عنوان نام دو کلاس برای دستهبندی تصاویر استفاده شود. این کد میتواند به جای classes در عنوان تصاویر قرار گیرد.

در بخش آخر، این کد نمودارهایی از خطا و دقت مدل CNN را روی دادههای آموزشی و آزمون رسم میکند. این نمودارها میتوانند به ما نشان دهند که مدل CNN چقدر خوب یاد گرفته است و چقدر به دادههای آموزشی

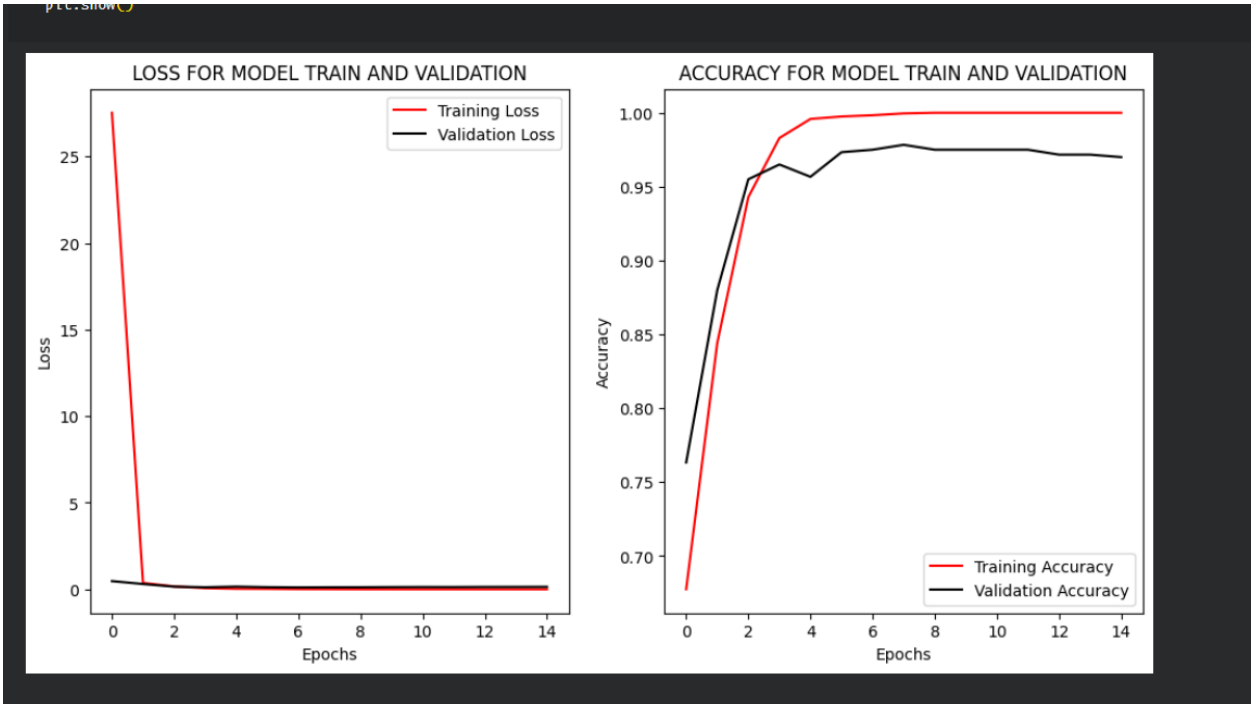
وابسته است. اگر خطا و دقت روی دادههای آموزشی و آزمون به هم نزدیک باشند، میتوانیم بگوییم که مدل CNN عمومی (generalized) است و روی دادههای جدید خوب عمل میکند. اما اگر خطا روی دادههای آزمون بیشتر و دقت روی دادههای آزمون کمتر از دادههای آموزشی باشد، میتوانیم بگوییم که مدل CNN بیشبرازش (overfitting) شده است و روی دادههای آموزشی حفظ کرده است. در این صورت، میتوانیم از روشهایی مانند افزایش دادهها (data augmentation)، افزودن لایههای حذف (dropout) یا تنظیم وزنها (weight regularization) برای بهبود عملکرد مدل CNN استفاده کنیم.

Plot loss and accuracy

In this part you should plot the loss and accuracy of train and test.

```
# plot training and test loss
plt.figure(figsize=(12,6))

plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='Training Loss', color ='red')
plt.plot(history.history['val_loss'], label='Validation Loss', color ='black')
plt.title('LOSS FOR MODEL TRAIN AND VALIDATION')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

# plot training and test accuracy
plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'], label='Training Accuracy' , color ='red')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy' , color ='black')
plt.title('ACCURACY FOR MODEL TRAIN AND VALIDATION')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

plt.show()
```

5]



تحلیل نمودار ها

مقدار لاس کاهشی بوده و نزولی شده و دقت مدل برای داده اموزش و تست قابل قبول بوده است و اختلاف کمی دارند مقدار دقت و مدل به خوبی تعمیم داده و در ولیدیشن با روندی که جلو می رود می تواند به بیش برازش و فیت شدن برسد

پیاده سازی فانکشنال

2.Functional API

Build a model

```
inputs_model = tf.keras.Input(shape=(256, 256, 1))
x = Conv2D(16, (3, 3), activation='relu')(inputs_model)
x = MaxPool2D((2, 2))(x)
x = Conv2D(32, (3, 3), activation='relu')(x)
x = MaxPool2D((2, 2))(x)
x = Conv2D(64, (3, 3), activation='relu')(x)
x = MaxPool2D((2, 2))(x)
x = Flatten()(x)
x = Dense(64, activation='relu')(x)
outputs_model = Dense(1, activation='sigmoid')(x)

model_func = tf.keras.Model(inputs=inputs_model, outputs=outputs_model)
model_func.summary()
```

ین کد یک شبکه عصبی پیچشی (CNN) را با استفاده از ساختار تابعی (functional) از کتابخانه TensorFlow ایجاد میکند. ساختار تابعی به ما امکان میدهد که یک مدل را با استفاده از لایههایی که به عنوان توابع از ورودی به خروجی اعمال میشوند، بسازیم. این ساختار دارای مزایایی مانند انعطافپذیری بیشتر، امکان ایجاد مدلهای چند ورودی و چند خروجی و امکان استفاده مجدد از لایهها و مدلها است.

لایه ورودی را تعریف می کنیم و به لایه های بعدی و میانی و غیره بهم وصل می کنیم و لایه خروجی ب لایه وسط وصل می کنم

مدل CNN در این کد شامل لایههای زیر است:

- لایه ورودی: این لایه شکل تصاویر ورودی را مشخص میکند. در این مثال، تصاویر ورودی دارای ابعاد ۲۵۶ در ۲۵۶ پیکسل و یک کانال رنگی هستند.

• لایههای پیچشی: این لایهها با استفاده از فیلترهایی با اندازه ۳ در ۳ پیکسل، ویژگیهای مهم تصاویر را استخراج میکنند. تعداد فیلترها در هر لایه پیچشی به ترتیب ۱۶، ۳۲ و ۶۴ است. همچنین از تابع فعالسازی relu برای افزایش خطیسازی استفاده شده است.

• لایههای حداکثر تجمیع (MaxPool): این لایهها با استفاده از پنجرههایی با اندازه ۲ در ۲ پیکسل، اندازه تصاویر را کاهش میدهند و از ویژگیهای مهم آنها حفظ میکنند. این لایهها بعد از هر لایه پیچشی قرار دارند.

• لایه صافی (Flatten): این لایه تصاویر را از حالت دوبعدی به حالت یکبعدی تبدیل میکند تا بتوانند به لایههای تماما متصل (Dense) متصل شوند.

• لایههای تماما متصل: این لایهها با استفاده از یادگیری عمیق، روابط بین ویژگیهای تصاویر را یاد میگیرند. تعداد نورونها در این لایهها به ترتیب ۶۴ و ۱ است. از تابع فعالسازی relu برای لایه اول و از تابع فعالسازی sigmoid برای لایه آخر استفاده شده است. تابع فعالسازی sigmoid مقدار خروجی را بین ۰ و ۱ قرار میدهد که میتواند به عنوان احتمال تعلق به یکی از دو کلاس تفسیر شود.

مدل CNN را میتوانید در شکل زیر مشاهده کنید.

```
tf.keras.utils.plot_model(model_func, show_shapes=True)
```

```
Model: "model_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_9 (InputLayer)        [(None, 256, 256, 1)]     0

 conv2d_24 (Conv2D)          (None, 254, 254, 16)      160

 max_pooling2d_24 (MaxPooli  (None, 127, 127, 16)      0
 ng2D)

 conv2d_25 (Conv2D)          (None, 125, 125, 32)      4640

 max_pooling2d_25 (MaxPooli  (None, 62, 62, 32)        0
 ng2D)

 conv2d_26 (Conv2D)          (None, 60, 60, 64)        18496

 max_pooling2d_26 (MaxPooli  (None, 30, 30, 64)        0
 ng2D)

 flatten_8 (Flatten)         (None, 57600)             0

 dense_16 (Dense)            (None, 64)                3686464

...
Total params: 3709825 (14.15 MB)
Trainable params: 3709825 (14.15 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

```
model_func.compile(optimizer='adam',
                   loss='binary_crossentropy',
                   metrics=['accuracy'])
```

Train the model

```
history = model_func.fit(
    datasetTrain,
    epochs=15,
    validation_data=datasetTest
)
```

```
Epoch 1/15
38/38 [==============================] - 7s 95ms/step - loss: 8.9907 - accuracy: 0.6971 - val_loss: 0.4384 - val_accuracy: 0.7767
Epoch 2/15
38/38 [==============================] - 4s 87ms/step - loss: 0.3641 - accuracy: 0.8196 - val_loss: 0.2856 - val_accuracy: 0.8800
Epoch 3/15
38/38 [==============================] - 4s 106ms/step - loss: 0.1913 - accuracy: 0.9262 - val_loss: 0.1829 - val_accuracy: 0.9400
Epoch 4/15
38/38 [==============================] - 5s 103ms/step - loss: 0.0888 - accuracy: 0.9683 - val_loss: 0.1370 - val_accuracy: 0.9550
Epoch 5/15
38/38 [==============================] - 4s 87ms/step - loss: 0.0548 - accuracy: 0.9871 - val_loss: 0.1362 - val_accuracy: 0.9650
Epoch 6/15
38/38 [==============================] - 6s 140ms/step - loss: 0.0247 - accuracy: 0.9942 - val_loss: 0.1320 - val_accuracy: 0.9700
Epoch 7/15
38/38 [==============================] - 6s 140ms/step - loss: 0.0147 - accuracy: 0.9979 - val_loss: 0.1397 - val_accuracy: 0.9733
Epoch 8/15
38/38 [==============================] - 6s 140ms/step - loss: 0.0108 - accuracy: 0.9983 - val_loss: 0.1362 - val_accuracy: 0.9700
Epoch 9/15
38/38 [==============================] - 4s 87ms/step - loss: 0.0085 - accuracy: 0.9992 - val_loss: 0.1553 - val_accuracy: 0.9717
Epoch 10/15
38/38 [==============================] - 5s 132ms/step - loss: 0.0074 - accuracy: 0.9996 - val_loss: 0.1677 - val_accuracy: 0.9717
Epoch 11/15
38/38 [==============================] - 4s 86ms/step - loss: 0.0070 - accuracy: 1.0000 - val_loss: 0.1720 - val_accuracy: 0.9733
Epoch 12/15
38/38 [==============================] - 5s 124ms/step - loss: 0.0073 - accuracy: 0.9996 - val_loss: 0.1769 - val_accuracy: 0.9733
Epoch 13/15
...
Epoch 14/15
38/38 [==============================] - 4s 87ms/step - loss: 0.0065 - accuracy: 0.9996 - val_loss: 0.1715 - val_accuracy: 0.9733
Epoch 15/15
38/38 [==============================] - 4s 97ms/step - loss: 0.0067 - accuracy: 0.9996 - val_loss: 0.1668 - val_accuracy: 0.9767
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

Test the model

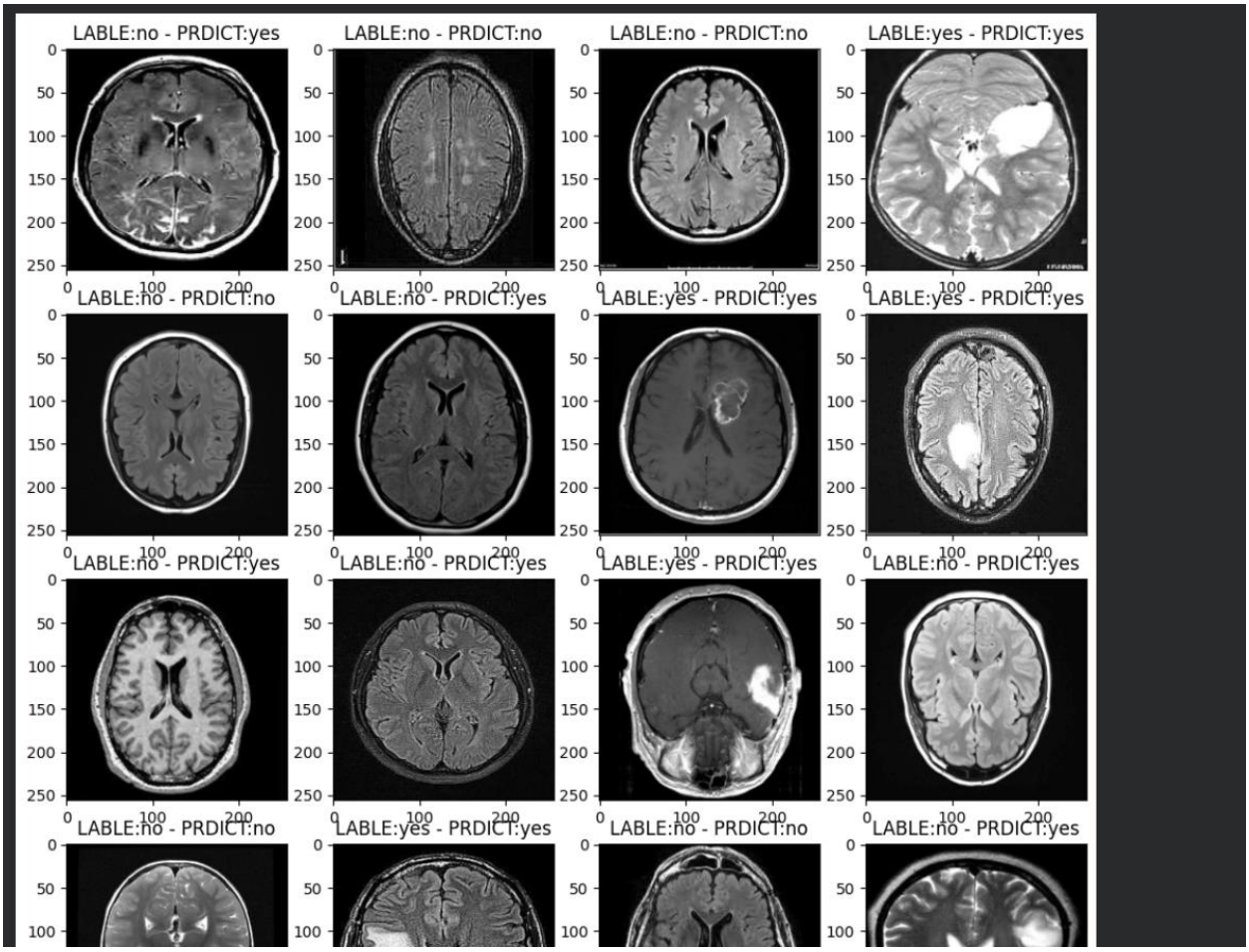
```
num_1 = model_func.evaluate(datasetTrain)
a= num_1[0]
b=num_1[1]
print('LOSS_TRAIN & ACCURACY_TRAIN', a , b )
num_2 = model_func.evaluate(datasetTest)
c= num_2[0]
d=num_2[1]
print('LOSS_TEST & ACCURACY_TEST', c , d )
```

```
38/38 [==============================] - 3s 53ms/step - loss: 0.0074 - accuracy: 1.0000
LOSS_TRAIN & ACCURACY_TRAIN 0.007442854810506105 1.0
10/10 [==============================] - 1s 53ms/step - loss: 0.1668 - accuracy: 0.9767
LOSS_TEST & ACCURACY_TEST 0.16676761209964752 0.9766666889190674
```

```
classes = ['yes', 'no']
imgg, labels = next(datasetTrain.take(1).as_numpy_iterator())
y = model_seq.predict(imgg)
fig = plt.figure(figsize=(12, 12))

for i in range(16):

  img, lb, pr = imgg[i], labels[i], y[i]
  j=i+1
  fig.add_subplot(4, 4, j)
  plt.imshow(img, cmap='gray')
  a= int(lb[0])
  b=int(pr[0])
  plt.title(f'LABLE:{classes[a]} - PRDICT:{classes[b]}')
```

```
2/2 [==============================] - 0s 10ms/step
```

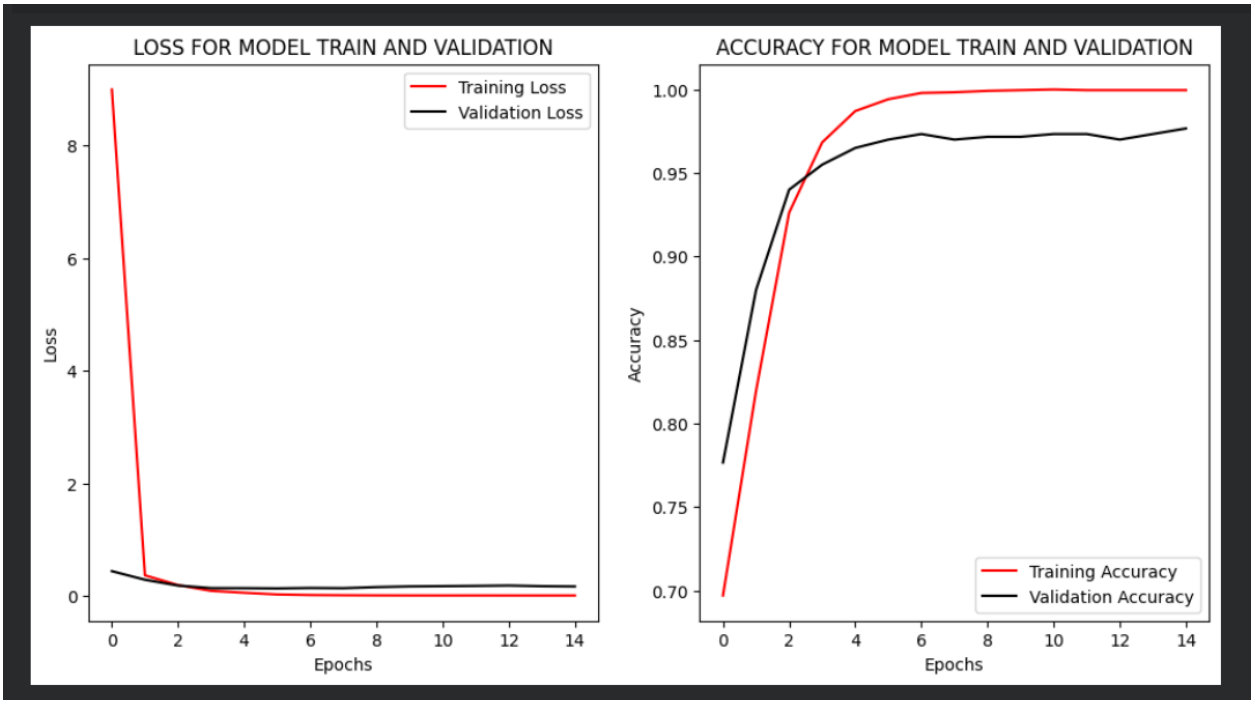
در اینجا دقت به اندازه ۰,۰۰۶۷ بیش تر است


LABLE:no - PRDICT:yes
LABLE:no - PRDICT:no
LABLE:no - PRDICT:no
LABLE:yes - PRDICT:yes
LABLE:no - PRDICT:no
LABLE:no - PRDICT:yes
LABLE:yes - PRDICT:yes
LABLE:yes - PRDICT:yes
LABLE:no - PRDICT:yes
LABLE:no - PRDICT:yes
LABLE:yes - PRDICT:yes
LABLE:no - PRDICT:yes
LABLE:no - PRDICT:no
LABLE:yes - PRDICT:yes
LABLE:no - PRDICT:no
LABLE:no - PRDICT:yes

Plot loss and accuracy

In this part you should plot the loss and accuracy of train and test.

```
# plot training and test loss
plt.figure(figsize=(12,6))

plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='Training Loss', color ='red')
plt.plot(history.history['val_loss'], label='Validation Loss', color ='black')
plt.title('LOSS FOR MODEL TRAIN AND VALIDATION')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

# plot training and test accuracy
plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'], label='Training Accuracy' , color ='red')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy' , color ='black')
plt.title('ACCURACY FOR MODEL TRAIN AND VALIDATION')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

plt.show()
```



سوال ۵)

یک مثال عملی از کاربرد لایههای همگشتی در دستهبندی تصاویر، میتواند شناسایی چهره باشد. در این مسئله، هدف این است که از یک تصویر ورودی، چهره های موجود در آن را تشخیص داده و با یک برچسب مشخص کنیم. برای این کار، میتوان از یک شبکه عصبی همگشتی استفاده کرد که از چندین لایه همگشتی، لایه های تاممتصل و لایه های خروجی تشکیل شده است. لایه های همگشتی میتوانند ویژگیهای مهم چهره ها را از تصویر استخراج کنند، مانند چشمها، بینی، دهان، ابروها و غیره. این ویژگیها میتوانند به شبکه عصبی کمک کنند که چهره های مختلف را از هم تمایز دهد و با دقت بالاتری دسته بندی کند. ویژگیهای منحصر به فرد لایه های همگشتی که منجر به عملکرد مفیدی میشوند، عبارتند از:

- لایه های همگشتی میتوانند از تغییرات جزئی در موقعیت، اندازه، رنگ و زاویه چهره ها در تصویر بیتفاوت باشند و به طور موثر ویژگیهای مربوط به چهره را شناسایی کنند.
- لایه

های همگشتی میتوانند از تکرار و ترکیب ویژگیهای سادهتر، ویژگیهای پیچیده تری را ایجاد کنند که برای تشخیص چهره ها مفیدتر هستند

- لایه های همگشتی میتوانند تعداد پارامترهای شبکه عصبی را کاهش دهند و از بیشبرازش جلوگیری کنند، زیرا از یک فیلتر همگشتی برای تمام نقاط تصویر استفاده میکنند و به اشتراک میگذارند.

به طور عکس، ممکن است در برخی حالات، ویژگیهای لایه های همگشتی چالشهایی را ایجاد کنند. برای مثال:

- لایه های همگشتی ممکن است از اطلاعات مکانی و هندسی چهرهها در تصویر از دست بدهند، زیرا از عملیاتی مانند ادغام (pooling) یا فلت کردن (flattening) استفاده میکنند که باعث کاهش ابعاد تصویر میشوند.
- لایه های همگشتی ممکن است با تصاویری که دارای نویز، تاری، تاریکی یا روشنایی زیاد هستند، مشکل داشته باشند و نتوانند ویژگیهای چهره ها را به خوبی استخراج کنند.
- لایه های همگشتی ممکن است با تصاویری که دارای چهره هایی با حالتهای مختلف هستند، مانند خنده، عصبانیت، تعجب و غیره، مشکل داشته باشند و نتوانند چهره ها را به درستی دسته بندی کنند.

تأثیر این چالشها بر کارایی مدل میتواند منفی باشد و باعث کاهش دقت، صحت و احتمال قطعیت شبکه عصبی شود. برای رفع این چالشها، میتوان از روشهایی مانند پیشپردازش تصاویر، افزایش دادهها، انتقال یادگیری، لایه های توجه یا شبکه های عصبی پیچشی بازگشتی استفاده کرد .

در اینجا برخی از کاربردهای رایج شبکه های عصبی کانولوشنال آورده شده است:

۱. تقسیم بندی معنایی: CNN ها می توانند هر پیکسل در یک تصویر را به کلاس های مختلف طبقه بندی کنند، به عنوان مثال. - انواع مختلف پوشش گیاهی در تصاویر ماهواره ای.

۲. تشخیص اشیا: CNN ها می توانند اشیاء درون یک تصویر را شناسایی کنند، به عنوان مثال. - شناسایی مکان و نوع وسیله نقلیه در جاده

۳. طبقه بندی تصاویر: CNN ها می توانند تصاویر را به دسته های مختلف طبقه بندی کنند، به عنوان مثال. شناسایی اشیاء در عکس

۴. شرح تصاویر: CNN ها می توانند توضیحاتی را به زبان طبیعی از تصاویر ایجاد کنند، به عنوان مثال. - توصیف اشیاء در یک عکس.

۵. تشخیص چهره - CNN ها می توانند هویت افراد مختلف را در تصاویر، مانند یافتن چهره افراد در فیلم های امنیتی، شناسایی و تأیید کنند.

۶. تجزیه و تحلیل تصویر پزشکی - CNN ها می توانند تومورها را در اسکن های پزشکی یا در تشخیص ناهنجاری ها در اشعه ایکس شناسایی کنند.

۷. تجزیه و تحلیل ویدئو - CNN ها می توانند حرکت اشیاء را در فریم ها تشخیص دهند.

۸. وسایل نقلیه خودمختار - CNN ها می توانند اشیاء را شناسایی و ردیابی کنند - مانند عابران پیاده و وسایل نقلیه دیگر.

https://www.engati.com/glossary/convolutional-neural-network

در اینجا مهمترین مزایای شبکه های عصبی کانولوشنال (CNN) آورده شده است:

۱. نیازی به نظارت انسانی نیست

۲. استخراج خودکار ویژگی

۳. بسیار دقیق در تشخیص و طبقه بندی تصویر

۴. تقسیم وزن

۵. محاسبات را به حداقل می رساند

۶. از دانش یکسانی در همه مکان‌های تصویر استفاده می‌کند.

۷. توانایی مدیریت مجموعه داده های بزرگ

۸. یادگیری سلسله مراتبی

اگرچه مزایای شبکه های عصبی کانولوشنال وجود دارد، اما معایبی نیز برای آن وجود دارد -

۱. الزامات محاسباتی بالا

۲. به داده های برچسب گذاری شده زیادی نیاز دارد

۳. ردپای حافظه بزرگ

۴. چالش های تفسیرپذیری

۵. اثربخشی محدود برای داده های متوالی

۶. تمایل به بسیار کندتر است , اموزش زمان بر است

Q1. کاربردهای CNN چیست؟

الف. شبکه های عصبی کانولوشنال (CNN) به دلیل استخراج ویژگی سلسله مراتبی در وظایف تجزیه و تحلیل تصویر و ویدئو برتری دارند. آنها کاربردهایی در تشخیص تصویر، تشخیص اشیا، تشخیص چهره، تجزیه و تحلیل تصویر پزشکی، خودروهای خودران و غیره پیدا می کنند. CNN ها از لایه های کانولوشنال و ادغام خود برای یادگیری خودکار ویژگی های مرتبط استفاده می کنند و آنها را در وظایف پردازش داده های بصری محوری می کند.

Q2. مزایا و کاربردهای CNN چیست؟

الف. شبکه های عصبی کانولوشنال (CNN) استخراج خودکار ویژگی ها را از تصاویر ارائه می دهند و پیش پردازش دستی را کاهش می دهند. آنها در کارهای مرتبط با تصویر مانند تشخیص اشیا، درک صحنه و تجزیه و تحلیل تصویر پزشکی برتری دارند. CNN ها می توانند سلسله مراتب فضایی را در داده ها به دلیل کانولوشن و لایه های ادغام خود ثبت کنند. این آنها را برای کارهایی که نیاز به تشخیص الگوی بصری پیچیده دارند ارزشمند می کند و زمینه هایی مانند بینایی رایانه و تجزیه و تحلیل تصویر مبتنی بر هوش مصنوعی را متحول کرده است.

Disadvantages:

Since convolutional neural networks are typically used for image-classification, we are generally dealing with high-dimensional data (images). While the structure of a ConvNet aims to mitigate over-fitting, you generally need a large amount of data for a convolutional neural network to work effectively. Of course, the amount of data you need depends on the complexity of the task at hand.

Related to the first point, convolutional neural networks are probably overkill if the task at hand is very simple (i.e. how can we distinguish between white vs. black circles?). For very simple tasks, you are better off doing basic processing on OpenCV since it's faster / efficient.

It takes a very long time to train a convolutional neural network, especially with large datasets. You generally need specialized hardware (like a GPU) to expedite the training process.

While CNNs are translation-invariant, they are generally bad at handling rotation and scale-invariance without explicit data augmentation.

Advantages:

CNN models are the golden standard for computer vision tasks since it does the feature extraction process for you. In the older days of computer vision, researchers used to manually extract features and implement classical ML algorithms (i.e. a simpler neural net, SVM) to do image classification tasks. However, CNN models generally perform superior compared to the manual extraction process (it's end-to-end)

کـاربرد های دیگر در مثال های عملی روزمره :

Medical imaging

In medical imaging, CNN is valuable in better accuracy in identifying tumours or other anomalies in X-ray and MRI images. Based on previously processed similar images by CNN networks, CNN models may analyse an image of a human body part, such as the lungs, and pinpoint where there might be a tumour and other anomalies like broken bones in X-ray images. Similarly, medical images like CT scans and mammograms can be used to diagnose cancer. In order to determine whether any indicators within a picture indicate malignancy or damage to cells owing to both hereditary and environmental factors, such as smoking habits, CNN models compare the image of a patient with database images that include comparable features.

Document analysis

Document analysis can also make use of convolutional neural networks. This has a significant impact on recognisers in addition to being helpful for handwriting analysis. A machine must process approximately a million commands per minute to scan someone's writing and compare it to its extensive database. By identifying words and phrases associated with the subject of a given document, CNN

networks can use both text and visuals to comprehend better what is written within.

Autonomous driving

Images can be modelled using convolutional neural networks (CNN), which are used to model spatial information. CNNs are regarded as universal non-linear function approximators because of their superior ability to extract features from images such as obstacles and interpret street signs. Furthermore, as the depth of the network grows, CNNs may detect a variety of patterns. For instance, the network's initial layers will record edges, but its deeper layers will capture aspects like an object's shape that are more complicated (leaves in trees or tyres on a vehicle). As a result, CNNs are the primary algorithm in self-driving cars.

Biometric authentication

By identifying specific physical traits connected to a person's face, CNN has been utilised for biometric identification of user identity. CNN models can be trained on people's images or videos to identify particular face traits like the space between the eyes, the nose's shape, the lips' curvature, etc. CNN models have also recognised various emotional states such as happiness or sadness based on photos or videos of people's faces. CNNs can also assess whether a subject is blinking in a photo and the general form of multiple-frame facial images.

منابع :

https://medium.com/swlh/convolutional-neural-network-expalained-in-7-real-life-examples-6015a64f9d2a

https://indiaai.gov.in/article/top-5-applications-of-convolution-neural-network

https://www.analyticsvidhya.com/blog/2021/10/applications-of-convolutional-/neural-networkscnn

.

سوال ۶)

الف)

هدف استفاده از فیلترهای ۱×۱ در شبکه های عصبی همگشتی این است که تعداد کانالها (channels) یا نقشه های ویژگی (feature maps) را کاهش دهند و به این ترتیب کاهش پیچیدگی و حافظه مورد نیاز شبکه را ایجاد کنند. فیلترهای ۱×۱ به عنوان یک لایه تغییر ابعاد (dimensionality reduction) عمل میکنند که هر کانال خروجی را به عنوان یک ترکیب خطی از کانال های ورودی محاسبه میکنند. این کار باعث میشود که ویژگیهای مهم ورودی حفظ شوند ولی تعداد پارامترها و محاسبات کاهش یابند.

برای مثال، اگر ورودی یک لایه همگشتی دارای ۲۵۶ کانال باشد و ما از یک فیلتر ۳×۳ با ۲۵۶ کانال استفاده کنیم، تعداد پارامترهای این لایه برابر با (۳×۳×۲۵۶×۲۵۶) + ۲۵۶ = ۵۹۰۰۸۰ خواهد بود. اما اگر ما ابتدا از یک فیلتر ۱×۱ با ۶۴ کانال استفاده کنیم و سپس از یک فیلتر ۳×۳ با ۲۵۶ کانال استفاده کنیم، تعداد پارامترهای این دو لایه برابر با (۱×۱×۲۵۶×۶۴) + ۶۴ + (۳×۳×۶۴×۲۵۶) + ۲۵۶ = ۶۹۹۵۲ خواهد بود. این یعنی تقریباً ۸۸ درصد کاهش در تعداد پارامترها.

لایه پیچشی ۱ X1برای "نمونه برداری از کانال متقاطع" یا ادغام کانال متقاطع استفاده شد .به عبارت دیگر، ۱ X1 Convبرای کاهش تعداد کانال ها و در عین حال معرفی غیر خطی استفاده شد .در ۱ X1 Convolutionبه سادگی به این معنی است که فیلتر دارای اندازه ۱ X1است) بله - این به معنای یک عدد واحد است که برخلاف ماتریسی مانند فیلتر ۳ .(X3این فیلتر ۱ X1بر روی کل تصویر ورودی پیکسل به پیکسل جمع می شود.

. کاهش/افزایش ابعاد

۲. کاهش بار محاسباتی با کاهش نقشه پارامتر

۳. اضافه کردن غیر خطی اضافی به شبکه

۴. از طریق لایه "Bottle-Neck" شبکه عمیق تری ایجاد کنید

۵. شبکه CNN کوچکتر ایجاد کنید که دقت بالاتری را حفظ کند

اگر بخواهیم عمق را کاهش دهیم و ارتفاع X Width نقشه‌های ویژگی (فیلد گیرنده) را ثابت نگه داریم، می‌توانیم فیلترهای ۱X (به یاد داشته باشید تعداد فیلترها = کانال‌های خروجی) را برای رسیدن به این اثر انتخاب کنیم. این اثر نمونه برداری متقابل کانالی را "کاهش ابعاد" می نامند.

لایه کانولوشن هم فیلد ادراکیش تمام تصویر/فیچرمپ ورودی هست. یعنی به کل ورودی متصل هست هر نورون . نه به نورون جلوییش.
هر نورن یک فیچرمپ تولید میکنه. برای اینکه عملیات بسرعت و خیلی بهینه پیاده سازی بشه بجای یک نورون چند کپی از اون رو ایجاد میکنن . همونطور که میدونید همه نورون های موجود در یه برش ستونی دارای وزن یکسان هستن و همه هم یه تابع غیرخطی روشون اعمال میشه . این یعنی همه یک نورون هستن که صرفا برای داشتن پیاده سازی بهینه به این شکل پیاده سازی شدن. از فیلترهای ۱*۱ در شبکه های همگشتی برای کاهش ابعاد در عمق استفاده می شود

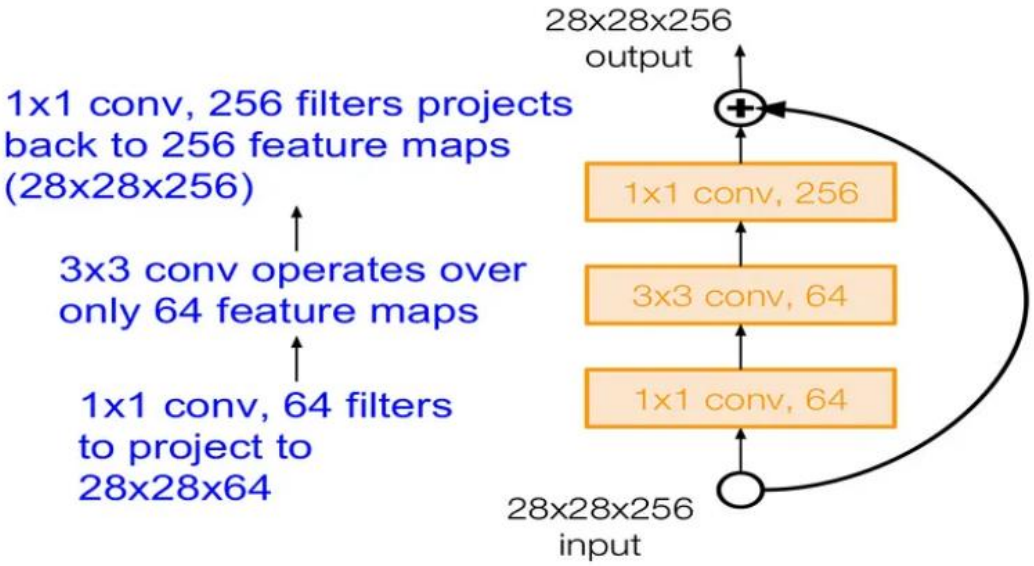
ب)

پس از اعمال فیلتر ۱×۱، نقشه ویژگی میتواند اطلاعات زیر را ارائه دهد:

- نقشه ویژگی میتواند تعداد کانالهای ورودی را کاهش دهد و به این ترتیب پیچیدگی و حافظه مورد نیاز شبکه را کاهش دهد. برای مثال، اگر ورودی دارای ۲۵۶ کانال باشد و ما از یک فیلتر ۱×۱ با ۶۴ کانال استفاده کنیم، خروجی دارای ۶۴ کانال خواهد بود ./

- نقشه ویژگی میتواند ارتباط میان کانالهای ورودی را در یک نقشه ویژگی یکپیکسلی به دست آورد و به این ترتیب ترکیب خطی از کانالهای ورودی را ایجاد کند. این کار میتواند به حفظ ویژگیهای مهم ورودی کمک کند

- نقشه ویژگی میتواند به عنوان یک لایه تماممتصل (fully-connected) عمل کند که هر نورون معادل با یک فیلتر ۱×۱ است. این کار میتواند به افزایش توانایی یادگیری شبکه کمک کند
./https://howsam.org/convolutional-neural-network



که چگونه یک لایه گردن بطری با استفاده از دنباله ای از ۳ لایه کانولوشن با فیلترهایی به اندازه X1۱، X3۳ و به دنبال آن X1۱ به ترتیب برای کاهش و بازیابی ابعاد طراحی شده است. نمونه برداری پایین از ورودی در لایه X1۱ اتفاق می افتد، بنابراین بردارهای ویژگی کوچکتر (تعداد پارامترهای کاهش یافته) برای تبدیل X3۳ به کار می رود. بلافاصله پس از آن لایه X1۱ ابعاد را برای مطابقت با بعد ورودی بازیابی می کند، بنابراین می توان از میانبرهای هویت به طور مستقیم استفاده کرد.

س از اعمال فیلترهای x1۱، یک نسخه تبدیل شده از نقشه ویژگی ورودی با کاهش یافته است

ابعاد هر پیکسل در خروجی مربوط به مجموع وزنی پیکسل های ورودی در آن است

میدان گیرنده، و این فرآیند به حفظ ویژگی‌های مهم کمک می‌کند و در عین حال ویژگی‌های کمتر مرتبط را کنار می‌گذارد

با لحاظ کردن عمق ویژگی ها واعمال این فیلتر روی هر پیکسل ترکیب ان ها ویژگی های جدید پیچیده تر را تولید کند که حاصل از ترکیب ویژگی های قبلی می باشد بنابر این ادغام و کوچک کردن و خلاصه کردن دیتا را در این فیچر میشود/

پ)

، همانطور که می‌دانید، ادغام کانال‌های متقابل را انجام می‌دهند - کانال‌ها را ترکیب می‌کند، اما 1X1فیلترهای ۱ نمی‌تواند ساختارهای فضایی را تشخیص دهد (به دلیل کار بر روی پیکسل‌های مجزا در مقابل یک وصله ورودی ساختارهای فضایی را تشخیص می دهد.3X3 Convolutionمانند فیلترهای بزرگتر). ۳

The 1X1 filters, as you know, performs cross channel pooling — Combines channels, but cannot detect spatial structures (by virtue of working on individual pixels as opposed to a patch of input like larger filters). The 3X3 Convolution detects spatial structures.

A 1×1 filter and a 3×3 filter are two types of convolutional filters that can be used in convolutional neural networks (CNNs). The main difference between them is the size of the filter, which affects the receptive field and the number of parameters of the layer. A 1×1 filter has a size of 1×1 pixels, whereas a 3×3 filter has a size of 3×3 pixels. Some of the advantages and disadvantages of using 1×1 and 3×3 filters are:

•A 1×1 filter can be used to reduce the number of channels or feature maps in a layer, which can reduce the complexity and memory requirements of the network. A 1×1 filter can also act as a linear projection of the input feature maps, which can preserve the important features while reducing the dimensionalityhttps.

•A 3×3 filter can be used to capture more spatial information and local patterns in the input feature maps, which can improve the performance and accuracy of the network. A 3×3 filter can also create more complex and non-linear features by combining and repeating simpler featureshttps

•A 1×1 filter has fewer parameters than a 3×3 filter, which can reduce the risk of overfitting and the computational cost of the network. However, a 1×1 filter may also lose some information and features that a 3×3 filter can capturehttps:

•A 3×3 filter has more parameters than a 1×1 filter, which can increase the expressive power and the learning capacity of the network. However, a 3×3 filter may also increase the risk of overfitting and the computational cost of the networkhttps

In practice, 1×1 and 3×3 filters can be used together in different combinations and architectures to achieve different goals and trade-offs. For example, some popular CNN models that use 1×1 and 3×3 filters are.

نقشه ویژگی همگشتی (convolutional feature map) یک ماتریس دوبعدی است که نتیجه اعمال یک فیلتر همگشتی (convolutional filter) بر روی تصویر اصلی یا نقشه ویژگی دیگر است. نقشه ویژگی همگشتی میتواند ویژگیهای مهم تصویر را استخراج کند، مانند لبهها، گوشهها، الگوها و غیره. نقشه ویژگی همگشتی از تصویر اصلی یا نقشه ویژگی دیگر با اندازههای مختلف متفاوت است، زیرا:

- نقشه ویژگی همگشتی معمولاً کوچکتر از تصویر اصلی است، زیرا فیلتر همگشتی ممکن است از حاشیههای تصویر فاصله بگیرد یا با یک گام (stride) بزرگتر از یک پیکسل حرکت کند
- نقشه ویژگی همگشتی ممکن است بزرگتر یا کوچکتر از نقشه ویژگی دیگر باشد، بسته به اندازه و گام فیلتر همگشتی که بر روی آن اعمال شده است فیچر ها : فیچر ها در قسمت های مختلف شبکه می تواند چیزهای گوناگونی اعم از لبه ها و بافت ها و .. باشد
- نقشه ویژگی همگشتی ممکن است دارای تعداد کانالهای (channels) متفاوتی با تصویر اصلی یا نقشه ویژگی دیگر باشد، بسته به تعداد فیلترهای همگشتی که بر روی آنها اعمال شدهاند
- نقشه ویژگی همگشتی ممکن است دارای مقادیر متفاوتی با تصویر اصلی یا نقشه ویژگی دیگر باشد، زیرا فیلتر همگشتی ممکن است از توابع فعالسازی (activation functions) مانند ReLU، Sigmoid یا Tanh استفاده کند که مقادیر را تغییر می دهند

فشرده شده کانال های اصلی ما می باشد که ویژگی های مهم را نگه داشته پرا متر را کم کرده و یک عدد واحد اسکالر مجزا از مقادیر اطراف و مختص به یک نقطا را می دهد.

بعد و عمق :

که توسط فیلتر های 1* 1قابل کنترل می باشد

ابعاد :

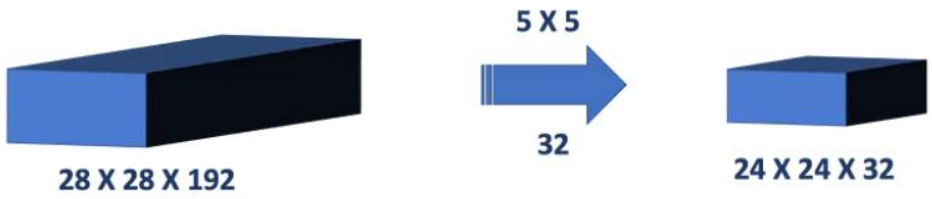
- نقشه ویژگی همگشتی ممکن است بزرگتر یا کوچکتر از نقشه ویژگی دیگر باشد، بسته به اندازه و گام فیلتر همگشتی که بر روی آن اعمال شده است فیچر ها : فیچر ها در قسمت های مختلف شبکه می تواند چیزهای گوناگونی اعم از لبه ها و بافت ها و .. باشد

فیچر ها :

نقشه ویژگی همگشتی (convolutional feature map) یک ماتریس دوبعدی است که نتیجه اعمال یک فیلتر همگشتی (convolutional filter) بر روی تصویر اصلی یا نقشه ویژگی دیگر است. نقشه ویژگی همگشتی میتواند ویژگیهای مهم تصویر را استخراج کند، مانند لبهها، گوشهها، الگوها و غیره. نقشه ویژگی همگشتی از تصویر اصلی یا نقشه ویژگی دیگر با اندازههای مختلف متفاوت است

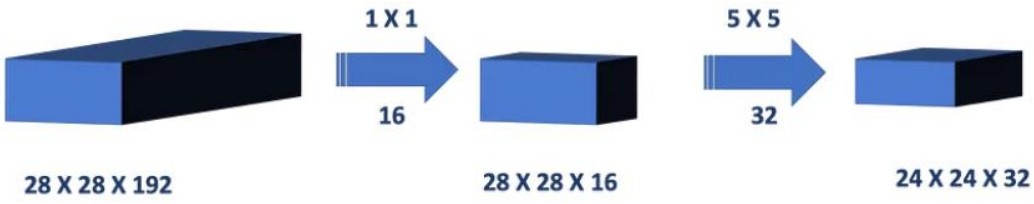
فیلتر ها و کرنل ها :

فیلتر ها و کرنا=ل هایی با سایز های گوناگون داریم که با توجه به اندازه اینپوت ساختار تغییر خواهند کرد .



Number of Operations : (28X28X32) X (5X5X192) = 120.422 Million Ops

Let us do some math with the same input feature maps but with 1X1 Conv layer before the 5 X 5 conv layer



Number of Operations for 1 X 1 Conv Step : (28X28X16) X (1X1X192) = 2.4 Million Ops
Number of Operations for 5 X 5 Conv Step : (28X28X32) X (5X5X16) = 10 Million Ops
Total Number of Operations = 12.4 Million Ops

ت)

وقتی شروع به بررسی بیشتر معماری‌های موفق مدرن CNN، مانند GoogleNet، ResNet و SqueezeNet می‌کنید، با لایه Convolution 1X1 مواجه می‌شوید که نقش اصلی را ایفا می‌کند. در نگاه اول، استفاده از یک رقم برای در هم تنیدگی با تصویر ورودی بیهوده به نظر می‌رسد (بعد از اینکه فیلترهای گسترده‌تری مانند X3، X5 می‌توانند بر روی یک تکه تصویر به جای یک پیکسل در این مورد کار کنند). با این حال، کانولوشن X1 ثابت کرده است که ابزار بسیار مفیدی است و به درستی به کار گرفته می‌شود، در ایجاد معماری‌های عمیق فوق‌العاده مؤثر خواهد بود.

فیلتر ۱×۱ در مدلهای همگشتی معروف یک نوع فیلتر همگشتی است که اندازه آن ۱×۱ است و میتواند تعداد کانالهای ورودی و خروجی را تغییر دهد. این فیلتر معمولاً به عنوان یک لایه تغییر ابعاد (dimensionality reduction) یا یک لایه تمام‌متصل (fully-connected) در مدلهای همگشتی استفاده میشود. برخی از مدلهای همگشتی معروف که از فیلتر ۱×۱ استفاده کرده‌اند عبارتند از:

- مدل GoogLeNet یا Inception که در سال ۲۰۱۴ معرفی شد و برنده مسابقه ImageNet شد. این مدل از فیلتر ۱×۱ برای کاهش تعداد پارامترها و محاسبات در لایه‌های Inception استفاده کرد
- مدل ResNet یا شبکه بازگشتی که در سال ۲۰۱۵ معرفی شد و نیز برنده مسابقه ImageNet شد. این مدل از فیلتر ۱×۱ برای تطبیق تعداد کانالها در لایه‌های اتصال بازگشتی (residual connection) استفاده کرد
- مدل MobileNet یا شبکه موبایل که در سال ۲۰۱۷ معرفی شد و برای کاربردهای موبایل و تعبیه شده طراحی شد. این مدل از فیلتر ۱×۱ برای ایجاد لایه‌های همگشتی نقطه‌ای (pointwise convolution) استفاده کرد

ث)

بله، حالتهایی وجود دارد که استفاده از فیلترهای ۱×۱ ممکن است مفید نباشد. برخی از این حالتها عبارتند از:

- اگر تعداد کانالهای ورودی و خروجی بسیار کم باشد، استفاده از فیلترهای ۱×۱ ممکن است باعث کاهش اطلاعات و ویژگیهای مهم ورودی شود و به جای کاهش پیچیدگی، باعث افزایش خطا شود
- اگر تصویر ورودی دارای ویژگیهای غیرخطی و پیچیده باشد، استفاده از فیلترهای ۱×۱ ممکن است نتواند این ویژگیها را به خوبی یاد بگیرد و نیاز به فیلترهای با اندازه‌های بزرگتر داشته باشد

- اگر تصویر ورودی دارای نویز، تاری، تاریکی یا روشنایی زیاد باشد، استفاده از فیلترهای ۱×۱ ممکن است نتواند این عوامل را حذف یا کاهش دهد و نیاز به پیشپردازش تصویر یا استفاده از فیلترهای با اندازههای بزرگتر داشته باشد/.

- اگر نیاز به کاهش ابعاد نداشته باشیم و تعداد لایه ها و ویژگی میانی کم باشد انگاه فقط هزینه محاسباتی به شبکه کوچک خود اضافه کردیم و زمان اموزش افزایش پیدا کرده و جایی که نیاز به اطلاعات همسایگی از پیکسل های اطراف داشته باشم ان را حذف کنیم و دیتا از دست بدهیم و اورفیت شویم یا جایی که باید از کرنل های بزرگتر متناسب با اندازه مسئله استفاده کنیم

ج)

```python
import tensorflow as tf
from tensorflow.keras import layers, models
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, Input


input_shape = (32, 32, 128)


model_11 = Sequential()
model_11.add(Input(shape=(32, 32, 128)))
model_11.add(Conv2D(filters=32, kernel_size=(1, 1), activation='relu' ,  padding='same'))
model_11.add(Conv2D(filters=1, kernel_size=(1, 1), activation='relu' ,  padding='same'))
model_11.compile(optimizer='SGD', loss='sparse_categorical_crossentropy', metrics=['accuracy'])


model_11.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_5 (Conv2D)           (None, 32, 32, 32)        4128

 conv2d_6 (Conv2D)           (None, 32, 32, 1)         33

=================================================================
Total params: 4161 (16.25 KB)
Trainable params: 4161 (16.25 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
model_33 = Sequential()
model_33.add(Input(shape=(32, 32, 128)))
model_33.add(Conv2D(filters=32, kernel_size=(3, 3), activation='relu' ,  padding='same'))
model_33.add(Conv2D(filters=32, kernel_size=(3, 3), activation='relu' ,  padding='same'))
model_33.compile(optimizer='SGD', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

[21]

```
model_33.summary()
```

[22]

```
Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_9 (Conv2D)           (None, 32, 32, 32)        36896

 conv2d_10 (Conv2D)          (None, 32, 32, 32)        9248

=================================================================
Total params: 46144 (180.25 KB)
Trainable params: 46144 (180.25 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
input_ex = np.random.rand(1, 32, 32, 128).astype(np.float32)
output_ex_11 = model_11.predict(input_ex)
output_ex_33 = model_33.predict(input_ex)
```

[25]

```
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 25ms/step
```

```
print(f"Input size: {input_ex.shape}")
print(f"Output size with 1*1 kernel : {output_ex_11.shape}")
print(f"Output size without 1*1 kernel: {output_ex_33.shape}")
```

[26]

```
Input size: (1, 32, 32, 128)
Output size with 1*1 kernel : (1, 32, 32, 1)
Output size without 1*1 kernel: (1, 32, 32, 32)
```

همانگونه که میبینیم اندازه ورودی کاهش یافته همچنین تفاوت دو مدلی که از کرنل ۱*۱ استفاده کرده اند و نکرده اند معلوم است کرنل ۱ با ترکیب ویژگی ها کانالی با عمق کم تر تولید کرده است .

میبینیم که در پیش بینی کردن مدل هم کرنل با سایز ۱ سریع تر عمل می کند همچنین پارامتر های قابل اموزش و پارامتر های کرنل ۱ یک دهم کرنل ۳ است .

همچنین با استفاده از کرنل ۱ می توانیم برای افزایش عمق نقشه های ویژگی بدون تغییر بعد های فضایی استفاده نمود .

در نتیجه می بینیم عرض و طول کانال تغییری نکرده ولی تعداد کانال ها به دلیل استفاده از کرنل ۱ تبدیل به یک کانال شده و کاهش یافته از ۱۲۸ به ۱

توضیح کد :

برای توضیح کد از بینگ استفاده شده است

این کد یک برنامه پایتون است که با استفاده از کتابخانه تنسورفلو، دو مدل شبکه عصبی کانولوشنی را برای دستهبندی تصاویر ایجاد میکند. این مدلها از لایههای کانولوشنی با اندازههای مختلف فیلتر (۱۱ و ۳۳) استفاده میکنند و خروجی آنها را با یک ورودی تصادفی مقایسه میکنند. توضیحات بیشتر به شرح زیر است:

• ابتدا کتابخانههای مورد نیاز را وارد میکنیم. تنسورفلو یک کتابخانه محبوب برای یادگیری عمیق است که امکان ساخت و آموزش مدلهای پیچیده را فراهم میکند. کراس یک واسط بالا سطح برای تنسورفلو است که کار با لایهها، مدلها و توابع خسارت و بهینهسازی را آسانتر میکند. نامپای یک کتابخانه برای کار با آرایهها و محاسبات عددی است.

```
import tensorflow as tf
from tensorflow.keras import layers, models
import numpy as np
```

• سپس دو مدل شبکه عصبی کانولوشنی را با استفاده از کلاس Sequential از کراس میسازیم. این کلاس امکان اضافه کردن لایهها به صورت خطی را میدهد. هر مدل شامل یک لایه ورودی و دو لایه کانولوشنی است. لایه ورودی مشخص میکند که شکل ورودی چگونه است. در اینجا ورودی یک تصویر با ابعاد ۳۲۳۲ پیکسل و ۱۲۸ کانال رنگی است. لایههای کانولوشنی مسئول استخراج ویژگیهای مهم از تصویر هستند. این لایهها از یک فیلتر که روی تصویر حرکت میکند و با آن ضرب نقطهای میکند، تشکیل شدهاند. فیلترها میتوانند اندازههای مختلفی داشته باشند. در اینجا مدل اول از فیلترهای ۱۱ و مدل دوم از فیلترهای ۳*۳ استفاده میکند. تعداد فیلترها مشخص میکند که خروجی چند کانال خواهد داشت. در اینجا مدل اول در لایه اول ۳۲ کانال و در لایه دوم ۱ کانال و مدل دوم در هر دو لایه ۳۲ کانال خروجی میدهد. فعالسازی مشخص میکند که تابع غیرخطی که بر روی خروجی اعمال میشود چیست. در اینجا از تابع relu که مقادیر منفی را صفر میکند و مقادیر مثبت را حفظ میکند، استفاده شده است. padding مشخص میکند که آیا تصویر را با صفرهای اضافی در اطراف پر میکنیم یا خیر. در اینجا از حالت same که باعث میشود خروجی همان اندازه ورودی باشد، استفاده شده است.

```
(input_shape = (32, 32, 128

()model_11 = Sequential
model_11.add(Input(shape=(32, 32, 128)))
model_11.add(Conv2D(filters=32, kernel_size=(1, 1), activation='relu' ,
padding='same'))
model_11.add(Conv2D(filters=1, kernel_size=(1, 1), activation='relu' ,
padding='same'))
model_11.compile(optimizer='SGD', loss='sparse_categorical_crossentropy',
metrics=['accuracy'])

()model_11.summary

()model_33 = Sequential
model_33.add(Input(shape=(32, 32, 128)))
model_33.add(Conv2D(filters=32, kernel_size=(3, 3), activation='relu' ,
padding='same'))
model_33.add(Conv2D(filters=32, kernel_size=(3, 3), activation='relu' ,
padding='same'))
model_33.compile(optimizer='SGD', loss='sparse_categorical_crossentropy',
metrics=['accuracy'])

()model_33.summary
```

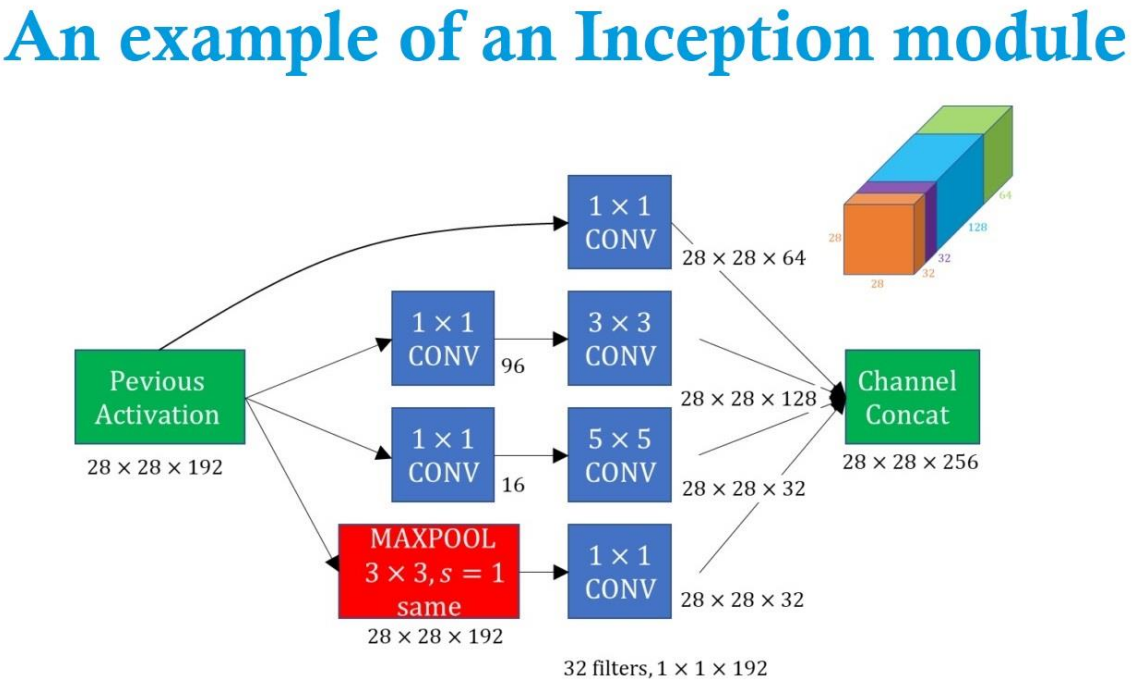
- بعد از ساخت مدلها، آنها را با یک ورودی تصادفی آزمایش میکنیم. این ورودی یک آرایه چهار بعدی است که ابعاد آن با شکل ورودی مدلها مطابقت دارد. مقادیر آرایه به صورت تصادفی از توزیع یکنواخت بین صفر و یک انتخاب شدهاند. متد predict مدلها را با این ورودی فراخوانی میکنیم و خروجی آنها را در متغیرهای output_ex_11 و output_ex_33 ذخیره میکنیم. این خروجیها نیز آرایههای چهار بعدی هستند که ابعاد آنها با تعداد کانالهای خروجی مدلها مطابقت دارد.

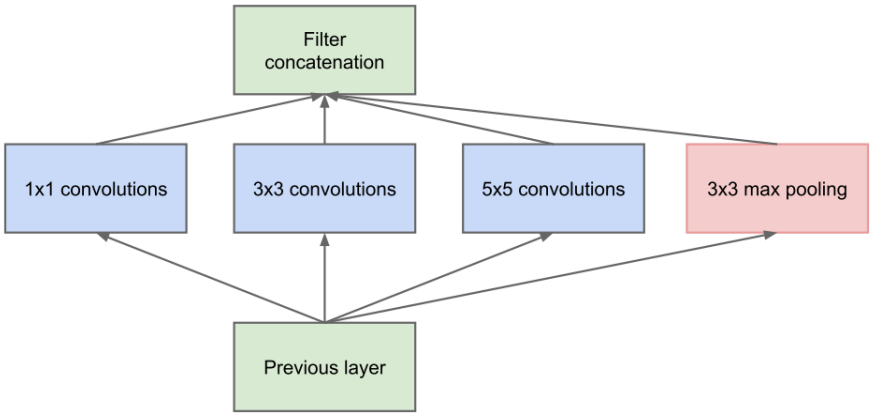
```
input_ex = np.random.rand(1, 32, 32, 128).astype(np.float32)
output_ex_11 = model_11.predict(input_ex)
output_ex_33 = model_33.predict(input_ex)
```

- در نهایت، اندازه ورودی و خروجیها را چاپ میکنیم. میبینیم که ورودی یک آرایه با ابعاد ۱۳۲۳۲۱۲۸ است. خروجی مدل اول یک آرایه با ابعاد ۱۳۲۳۲۱ است. خروجی مدل دوم یک آرایه با ابعاد ۱۳۲۳۲۳۲ است. این نشان میدهد که مدل اول با استفاده از فیلترهای ۱۱ تعداد کانالها را کاهش داده است و مدل دوم با استفاده از فیلترهای ۳*۳ تعداد کانالها را حفظ کرده است. این میتواند برای کاهش ویژگی ها مناسب باشد.

سوال ۷)

۱.





The purpose of using Inception blocks in convolutional models is to learn features in multiple scales and reduce the computational cost of the model. Inception blocks allow us to use multiple types of filter sizes, such as 1x1, 3x3, and 5x5, in a single block, instead of being restricted to a single filter size. This way, the model can capture both local and global patterns in the images. Moreover, Inception blocks use 1x1 convolutions to reduce the number of channels in the input data, which reduces the number of parameters and the complexity of the model. Inception blocks are designed to approximate an optimal local sparse structure in a CNN, which means that only a subset of the filters are relevant for each region of the input

Advantages of the Inception Module

The Inception Module offers several advantages over traditional CNN architectures:

Key Features of the Inception Module

The Inception Module is characterized by several key features that differentiate it from traditional CNN layers:

- **Multi-level Feature Extraction:** The module applies several convolutional filters of different sizes (e.g., 1x1, 3x3, 5x5) to the input simultaneously. This allows the network to capture information at various scales and complexities.

- **Dimensionality Reduction:** The use of 1x1 convolutions serves as a method for dimensionality reduction, reducing computational complexity and the number of parameters without losing depth in the network.
- **Pooling:**

  In addition to convolutional filters, the Inception Module includes a parallel pooling branch (usually max pooling), which provides another form of spatial aggregation.

- **Concatenation:** The outputs of all filters and the pooling layer are concatenated along the channel dimension before being fed to the next layer. This concatenation ensures that the subsequent layers can access features extracted at different scales.

- **Efficiency:** By implementing filters of multiple sizes, the module efficiently uses computing resources to extract relevant features without the need for deeper or wider networks.
- **Reduced Overfitting:** The architecture's complexity and depth help in learning more robust features, which can reduce overfitting, especially when combined with other regularization techniques.
- **Improved Performance:** Networks with Inception Modules have shown improved performance on various benchmark datasets for image recognition and classification tasks.

  The Inception Module represents a significant milestone in the development of CNNs for deep learning. Its innovative approach to multi-scale feature extraction has influenced the design of subsequent neural network architectures and has contributed to the advancement of state-of-the-art performance in computer vision tasks. As deep learning continues to evolve, the principles behind the Inception Module remain relevant for building efficient and powerful neural networks.

  Challenges with the Inception Module

  While the Inception Module brings many benefits, it also introduces certain challenges:

- **Increased Complexity:** The architecture of the Inception Module is more complex than traditional layers, which can make it harder to design and train.
- **Hyperparameter Tuning:**

The module introduces additional hyperparameters, such as the number and sizes of filters, which require careful tuning to achieve optimal performance.

- **Resource Intensity:** Although designed for efficiency, the Inception Module can still be resource-intensive due to the large number of operations and concatenation of outputs.

Evolution of the Inception Module

Since its introduction, the Inception Module has evolved through several iterations, leading to improved versions such as Inception-v2, Inception-v3, and Inception-v4. These versions have introduced various optimizations, including factorization of convolutions, expansion of the filter bank outputs, and the use of residual connections.

One notable variant is the Inception-ResNet hybrid, which combines the Inception architecture with residual connections from ResNet, another influential CNN architecture. This combination allows for even deeper networks by enabling more efficient training and better gradient flow.

Applications of Networks with Inception Modules

Convolutional neural networks that incorporate Inception Modules have been successfully applied to a wide range of computer vision tasks, including:

- Image classification
- Object detection
- Face recognition
- Image segmentation

These networks have been particularly impactful in situations where capturing multi-scale information is crucial for accurate predictions.

Conclusion

The Inception Module represents a significant milestone in the development of CNNs for deep learning. Its innovative approach to multi-scale feature extraction has influenced the design of subsequent neural network architectures and has contributed to the advancement of state-of-the-art performance in computer vision tasks. As deep learning continues to evolve, the principles behind the Inception Module remain relevant for building efficient and powerful neural networks.

https://deepai.org/machine-learning-glossary-and-terms/inception-module#:~:text=Inception%20Modules%20are%20used%20in,as%20overfitting%2C%20among%20other%20issues

https://arxiv.org/pdf/1409.4842v1.pdf

.

توضیحات تکمیلی

ویژگی های مختلف و گوناگون با هم ترکیب می شود و این ترکیب خروجی های مختلف می تواند اطلاعات بیش تر و پیچیده تر را از تصاویر استخراج کند. عملیات های کانولوشن به صورت پیوسته و موازی در شبکه عصبی اعمال می شود و با کانکریت کردن لایه ها به نتایج مطلوب میرسیم لایه ی 1*1 تعداد کانال ها را همانگونه که گفتیم تعداد کانال ها و ابعاد را کاهش می دهد و فیلتر های دیگر ۳*۳ , ۵*۵ و اعمال ماکس پولینگ که ابعاد فضای ورودی را کاهش داده و مدل را نسبت به مواردی مقاوم می کند می توان ویژگی های مختلف استخراج کرد و می توان تصاویر را در هر جای موقعیت مکانی و با اندازه های کوناگون دور و نزدیک شناسایی کرد و لایه ها به صورت موازی به استخراج ویژگی های مختلف بپردازند و تعمیم دهی خوبی برای شناسایی انواع تصاویر داشته باشند

توضیح پیاده سازی کد

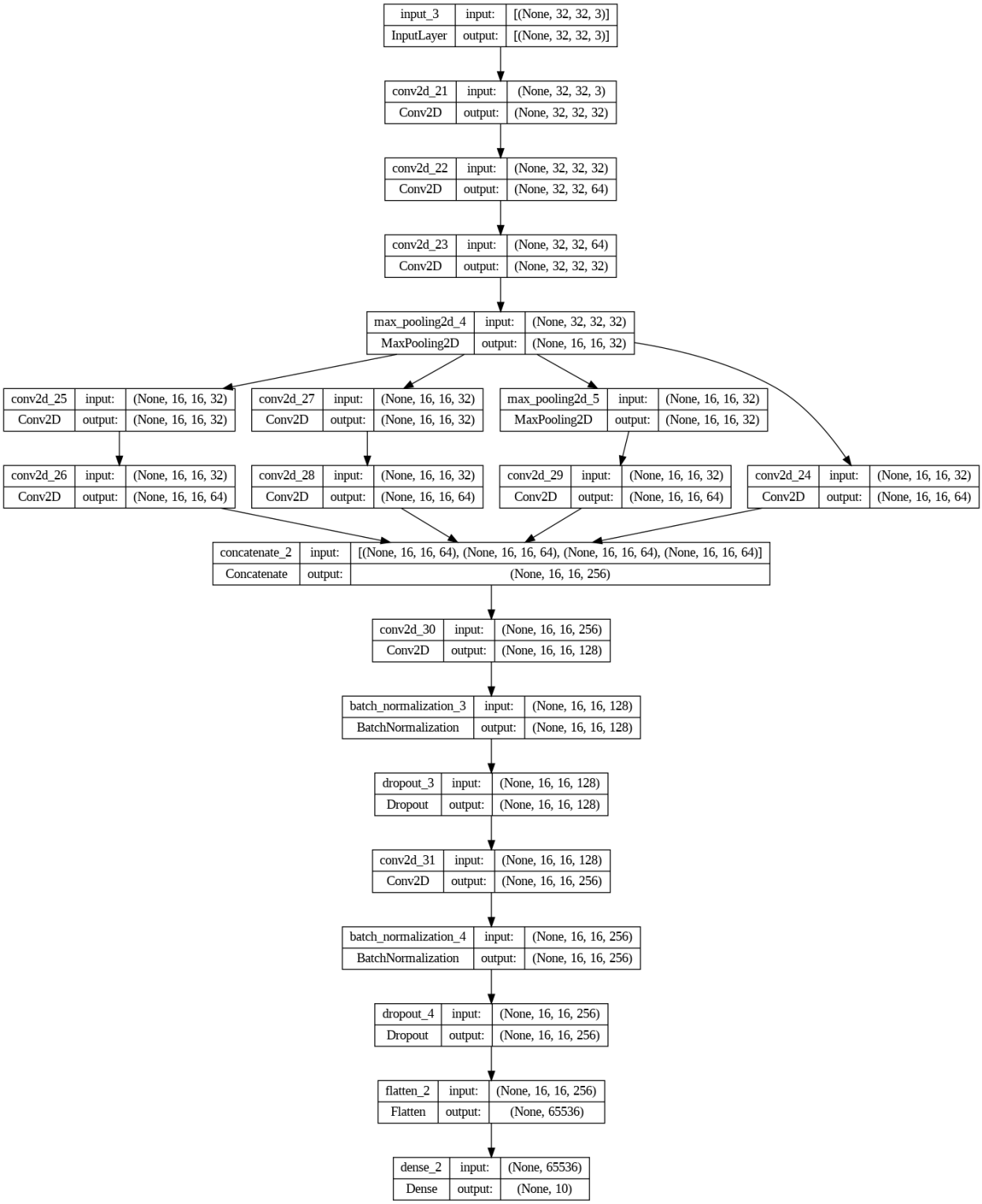
این کد یک مدل شبکه عصبی کانولوشنی با استفاده از کتابخانه کراس برای دسته بندی تصاویر مجموعه داده cifar10 است. این مجموعه داده شامل ۶۰۰۰۰ تصویر رنگی x32۳۲ پیکسلی است که به ۱۰ کلاس مختلف تقسیم شده اند. این کد از چندین لایه کانولوشنی، لایه های ادغام بیشینه، لایه های افت و باز، لایه های نرمال سازی دسته ای و یک لایه کاملا متصل با تابع فعال سازی softmax برای پیش بینی احتمال هر کلاس استفاده می کند. این کد همچنین از یک ماژول inception برای افزایش کارایی مدل استفاده می کند. ماژول inception یک ساختار شبکه است که از چندین فیلتر کانولوشنی با اندازه های مختلف و یک لایه ادغام بیشینه بر روی ورودی اعمال می کند و سپس خروجی های آن ها را به یکدیگر الحاق می کند. این کد نیز از تابع خطای crossentropy چند دسته ای و بهینه ساز adam برای آموزش مدل استفاده می کند. این کد نمودار مدل را نشان می دهد و همچنین نمودار تغییرات خطا و دقت مدل را برای داده های آموزشی و آزمایشی رسم می کند. این کد از داده های آموزشی و آزمایشی که در کتابخانه کراس موجود هستند استفاده می کند و آن ها را به

صورت نرمال شده برای مدل ورودی می دهد. این کد همچنین برچسب های داده ها را به صورت one-hot encoding تبدیل می کند. این کد برای ۵ دوره مدل را با اندازه دسته ۲۵۶ آموزش می دهد و دقت مدل را بر روی داده های آزمایشی ارزیابی می کند

اول، داده های تصویری را به صورت نرمال شده برای مدل ورودی می دهد. این کار با تقسیم هر پیکسل تصویر به ۲۵۵ انجام می شود. این کار باعث می شود که مقادیر پیکسل ها بین ۰ و ۱ قرار بگیرند و مقیاس رنگی را از ۰ تا ۲۵۵ به ۰ تا ۱ تغییر دهد. این کار می تواند به بهبود عملکرد مدل کمک کند.

• دوم، برچسب های داده ها را به صورت one-hot encoding تبدیل می کند. این کار باعث می شود که هر برچسب به یک بردار ۱۰ تایی تبدیل شود که در آن فقط یک عنصر مقدار ۱ دارد و بقیه مقدار ۰ دارند. این کار می تواند به مدل کمک کند تا خروجی های خود را به صورت احتمالاتی برای هر کلاس تولید کند. برای مثال، اگر برچسب یک تصویر ۳ باشد، بردار one-hot encoding آن به صورت [۰, ۰, ۰, ۱, ۰, ۰, ۰, ۰, ۰, ۰] خواهد بود..

توضیحات جزئی تر به همراه خروجی ها در فایل ۷ و ۷-۴ امده است و تصویر مدل هم در ادامه امده


input_3 | input: | [(None, 32, 32, 3)]
InputLayer | output: | [(None, 32, 32, 3)]
conv2d_21 | input: | (None, 32, 32, 3)
Conv2D | output: | (None, 32, 32, 32)
conv2d_22 | input: | (None, 32, 32, 32)
Conv2D | output: | (None, 32, 32, 64)
conv2d_23 | input: | (None, 32, 32, 64)
Conv2D | output: | (None, 32, 32, 32)
max_pooling2d_4 | input: | (None, 32, 32, 32)
MaxPooling2D | output: | (None, 16, 16, 32)
conv2d_25 | input: | (None, 16, 16, 32)
Conv2D | output: | (None, 16, 16, 32)
conv2d_27 | input: | (None, 16, 16, 32)
Conv2D | output: | (None, 16, 16, 32)
max_pooling2d_5 | input: | (None, 16, 16, 32)
MaxPooling2D | output: | (None, 16, 16, 32)
conv2d_26 | input: | (None, 16, 16, 32)
Conv2D | output: | (None, 16, 16, 64)
conv2d_28 | input: | (None, 16, 16, 32)
Conv2D | output: | (None, 16, 16, 64)
conv2d_29 | input: | (None, 16, 16, 32)
Conv2D | output: | (None, 16, 16, 64)
conv2d_24 | input: | (None, 16, 16, 32)
Conv2D | output: | (None, 16, 16, 64)
concatenate_2 | input: | [(None, 16, 16, 64), (None, 16, 16, 64), (None, 16, 16, 64), (None, 16, 16, 64)]
Concatenate | output: | (None, 16, 16, 256)
conv2d_30 | input: | (None, 16, 16, 256)
Conv2D | output: | (None, 16, 16, 128)
batch_normalization_3 | input: | (None, 16, 16, 128)
BatchNormalization | output: | (None, 16, 16, 128)
dropout_3 | input: | (None, 16, 16, 128)
Dropout | output: | (None, 16, 16, 128)
conv2d_31 | input: | (None, 16, 16, 128)
Conv2D | output: | (None, 16, 16, 256)
batch_normalization_4 | input: | (None, 16, 16, 256)
BatchNormalization | output: | (None, 16, 16, 256)
dropout_4 | input: | (None, 16, 16, 256)
Dropout | output: | (None, 16, 16, 256)
flatten_2 | input: | (None, 16, 16, 256)
Flatten | output: | (None, 65536)
dense_2 | input: | (None, 65536)
Dense | output: | (None, 10)

۲.

اندازه پارامتر stride مشخص میکند که فیلتر کانولوشنی چه قدمی را در حرکت روی داده ورودی بردارد. این قدم میتواند افقی، عمودی یا هر دو باشد. اندازه stride بر ابعاد فضایی نگاشت ویژگیها تأثیر میگذارد زیرا باعث میشود که تعداد نقاطی که فیلتر روی آنها اعمال میشود کاهش یابد. به طور کلی، اگر stride بزرگتر باشد، ابعاد نگاشت ویژگیها کوچکتر میشوند و بالعکس. این رابطه را میتوان با فرمول زیر بیان کرد:

$$(n h - k h + p h + s h - 1) / s h \times (n w - k w + p w + s w - 1) / s w \quad (۷,۳,۱)$$

که در آن $n h$ و $n w$ ابعاد داده ورودی، $k h$ و $k w$ ابعاد فیلتر کانولوشنی، $p h$ و $p w$ اندازه padding و $s h$ و $s w$ اندازه stride هستند. این فرمول نشان میدهد که اگر stride افزایش یابد، ابعاد نگاشت ویژگیها کاهش مییابند. برای مثال، اگر داده ورودی ۲۵۶ × ۲۵۶ پیکسل باشد و فیلتر کانولوشنی ۳ × ۳ پیکسل باشد، با اندازه stride 1، ابعاد نگاشت ویژگیها ۲۵۴ × ۲۵۴ پیکسل خواهد بود. اما اگر اندازه stride 2 باشد، ابعاد نگاشت ویژگیها ۱۲۷ × ۱۲۷ پیکسل خواهد بود. بنابراین، اندازه stride میتواند برای کنترل اندازه خروجی و کاهش پیچیدگی محاسباتی مدل استفاده شود.

$$Output\ size = Input\ size - Pool\ size\ Strides / \text{strides+1}$$

Strideپارامتری است که حرکت هسته یا فیلتر را در میان داده های ورودی مانند یک تصویر دیکته می کند. هنگام انجام عملیات پیچیدگی، گام تعیین می کند که فیلتر در هر مرحله چند واحد جابجا می شود. این تغییر می تواند افقی، عمودی یا هر دو باشد، بسته به پیکربندی گام.

به عنوان مثال، یک گام ۱ فیلتر را هر بار یک پیکسل حرکت می دهد، در حالی که گام ۲ آن را دو پیکسل حرکت می دهد. یک گام بزرگ‌تر، بعد خروجی کوچک‌تری ایجاد می‌کند و به‌طور مؤثر تصویر را کاهش می‌دهد.

اهمیت گام

انتخاب گام از چند جهت بر مدل تأثیر می گذارد:

اندازه خروجی: یک گام بزرگتر منجر به یک بعد فضایی خروجی کوچکتر می شود. این به این دلیل است که فیلتر با هر مرحله منطقه بزرگ تری از تصویر ورودی را پوشش می دهد، بنابراین تعداد موقعیت هایی که می تواند اشغال کند کاهش می یابد.

کارایی محاسباتی: افزایش گام می تواند بار محاسباتی را کاهش دهد. از آنجایی که فیلتر پیکسل های بیشتری را در هر مرحله جابه جا می کند، عملیات کمتری را انجام می دهد که می تواند فرآیندهای آموزش و استنتاج را سرعت بخشد.

میدان دید: گام بالاتر به این معنی است که هر مرحله از فیلتر، ناحیه وسیع تری از تصویر ورودی را در نظر می گیرد. این می‌تواند زمانی مفید باشد که مدل به جای تمرکز بر جزئیات دقیق‌تر، ویژگی‌های جهانی بیشتری را به تصویر بکشد.

نمونه برداری پایین:

گام‌ها را می‌توان به عنوان جایگزینی برای لایه‌های ادغام‌شده برای نمونه‌برداری پایین ورودی استفاده کرد. لایه‌های ادغام، مانندmax pooling ، اغلب برای کاهش ابعاد فضایی و ایجاد تغییر ناپذیری در ترجمه‌های کوچک استفاده می‌شوند. با این حال، افزایش گام در یک لایه کانولوشن می‌تواند به یک اثر مشابه بدون نیاز به لایه ادغام اضافی دست یابد.

در تمرین گام بردارید

در عمل، گام اغلب بر روی ۱ یا ۲ تنظیم می‌شود. زمانی که مدل نیاز به حفظ وضوح بالایی از ویژگی‌ها دارد، یک گام معمولی است، که به ویژه در لایه‌های اولیه شبکه مهم است. یک گام ۲ یا بیشتر ممکن است در لایه‌های عمیق‌تر یا زمانی که تصاویر ورودی بزرگ هستند استفاده شود و مدل باید ابعاد را کاهش دهد تا تعداد پارامترها و هزینه محاسباتی را کنترل کند.

توجه به این نکته مهم است که در حالی که افزایش گام می تواند کارایی محاسباتی را بهبود بخشد، ممکن است منجر به از دست دادن اطلاعات نیز شود. گام‌های بزرگ‌تر از ۱ از پیکسل‌ها عبور می‌کنند، که می‌تواند حاوی

اطلاعات مفیدی برای استخراج ویژگی باشد. بنابراین، انتخاب گام یک معامله است که باید بر اساس وظیفه و مجموعه داده خاص به دقت مورد توجه قرار گیرد.

محاسبه اندازه خروجی با گام

اندازه خروجی یک عملیات کانولوشن را می توان با استفاده از فرمول زیر محاسبه کرد:

$$O = ((W - K + 2P) / S) + 1$$

جایی که:

Oاندازه خروجی است

Wاندازه ورودی (عرض یا ارتفاع) است

Kاندازه هسته است

بالشتک است

Sگام است

این فرمول به تعیین ابعاد نقشه ویژگی خروجی کمک می کند، که برای طراحی و درک معماری یک CNN ضروری است.

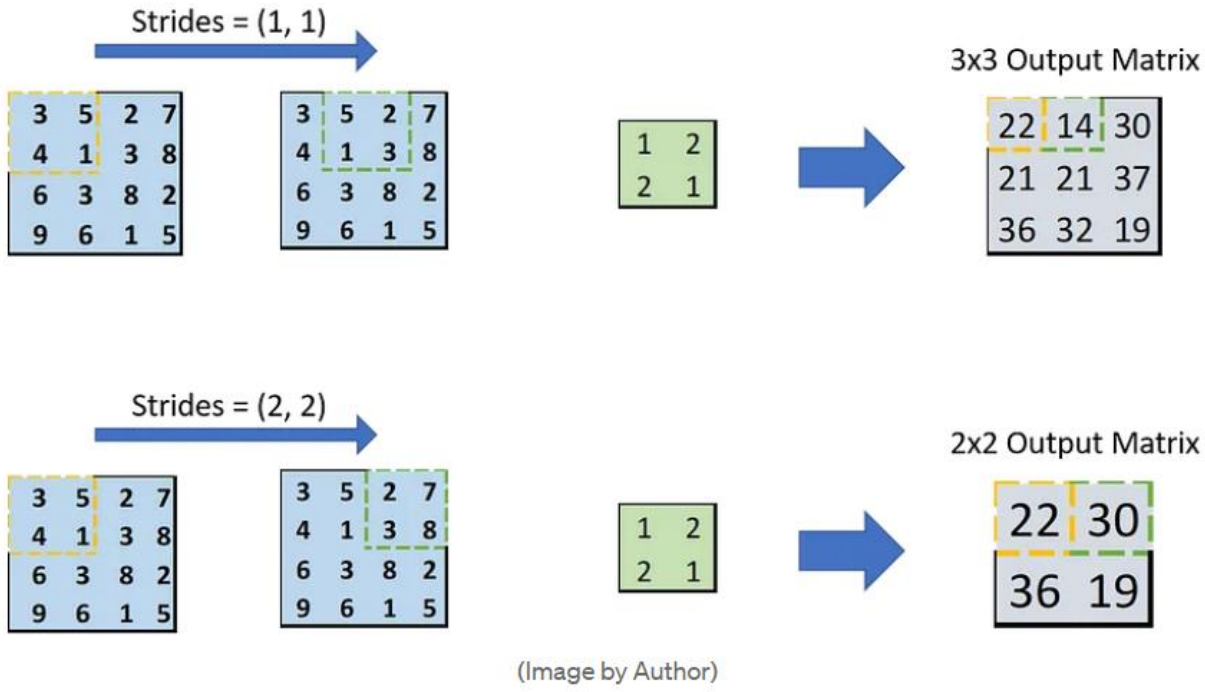
نتیجه

Strideیک فراپارامتر اساسی در شبکه های عصبی کانولوشن است که بر عملکرد و کارایی مدل تأثیر می گذارد. نحوه تعامل فیلترهای کانولوشنال با داده های ورودی را کنترل می کند و بر اندازه نقشه های ویژگی

خروجی تأثیر می گذارد. درک و انتخاب گام مناسب برای بهینه سازی CNN ها برای وظایف مختلف در تجزیه و تحلیل تصویر و ویدئو، و همچنین سایر حوزه هایی که CNN ها در آنها قابل اجرا هستند، بسیار مهم است.

هنگام طراحی یک شبکه عصبی کانولوشن، باید پیامدهای گام برداشتن بر توانایی شبکه برای گرفتن ویژگی‌های مرتبط، الزامات محاسباتی و عملکرد کلی مدل را در نظر گرفت. ایجاد تعادل بین این عوامل کلید توسعه CNN های موثر و کارآمد برای برنامه های یادگیری ماشین است.

In convolutions, the strides parameter indicates how fast the kernel moves along the rows and columns on the **input layer**. If a stride is (1, 1), the kernel moves one row/column for each step; if a stride is (2, 2), the kernel moves two rows/columns for each step. As a result, the larger the strides, the faster you reach the end of the rows/columns, and therefore the smaller the output matrix (if no padding is added). Setting a larger stride can also decrease the repetitive use of the same numbers.



https://towardsdatascience.com/understand-transposed-convolutions-and-build-your-own-transposed-convolution-layer-from-scratch-

4f5d97b2967#:~:text=In%20convolutions%2C%20the%20strides%20parameter,rows%2Fcolumns%20for%20each%20step.

۳.

لایه های مختلف استفاده شده :

Maxpooling 2d

Max pooling 2D is a type of layer that reduces the spatial dimensions of the input feature map by applying a max operation over a sliding window. The max operation selects the maximum value in each window and discards the rest. This can help to reduce the number of parameters and the computational cost of the model, as well as to enhance the translation invariance of the features. Max pooling 2D is often used after convolutional layers to downsample the feature maps and extract the most salient features.

The code layer= MaxPooling2D(pool_size=(2, 2))(layer) creates a max pooling 2D layer that takes the previous layer as the input and applies a max operation over a 2x2 window. The pool_size parameter specifies the size of the window that slides over the input. The output of this layer will have half the height and width of the input layer, and the same number of channels. For example, if the input layer has a shape of (32, 32, 64), the output layer will have a shape of.(٦٤ ,١٦ ,١٦)

Conv2D

Conv2D is a type of layer that performs a mathematical operation called convolution on the input data, which can be an image or a feature map. Convolution is a process of sliding a filter or a kernel over the input and computing the dot product of the filter and the input at

each position. This can help to extract features or patterns from the input data, such as edges, shapes, colors, etc. Conv2D is often used as the first layer of a convolutional neural network, which is a type of network that can learn to recognize and classify images.

The code layer = Conv2D(64, (3,3), padding='same', activation='relu')(layer) creates a Conv2D layer that takes the previous layer as the input and applies 64 filters of size 3x3 to it. The padding parameter is set to 'same', which means that the output layer will have the same spatial dimensions as the input layer. The activation parameter is set to 'relu', which means that the rectified linear unit function will be applied to the output of the convolution. This function returns the maximum of zero and the input value, and is commonly used to introduce non-linearity in neural networks. A Conv2D layer can have multiple filters, each of which can produce a different feature map. The output of a Conv2D layer is the concatenation of all the feature maps.

Inception module

An inception module is a type of network structure that can increase the efficiency and performance of a convolutional neural network by allowing it to learn features at multiple scales and levels of abstraction. The inception module consists of four parallel branches that apply different types of convolutional filters and a max pooling layer to the input feature map. The outputs of these branches are then concatenated along the channel dimension to form the output of the inception module. The inception module can help the model to capture both local and global features, as well as to reduce the number of parameters and the computational cost of the model.

The code you provided implements an inception module using the Keras library. The code defines a function that takes an input feature map x and returns an output feature map output_inception. The function does the following steps:

- First, it creates a branch that applies a 1x1 convolution with 64 filters and a relu activation function to the input x. This branch can help to reduce the dimensionality of the input and to learn linear features.
- Second, it creates a branch that applies a 1x1 convolution with 32 filters and a relu activation function to the input x, followed by a 3x3 convolution with 64 filters and a relu activation function. This branch can help to learn features at a larger spatial scale and to increase the non-linearity of the model.
- Third, it creates a branch that applies a 1x1 convolution with 32 filters and a relu activation function to the input x, followed by a 5x5 convolution with 64 filters and a relu activation function. This branch can help to learn features at an even larger spatial scale and to capture more complex patterns.
- Fourth, it creates a branch that applies a 3x3 max pooling layer with a stride of 1 and a same padding to the input x, followed by a 1x1 convolution with 64 filters and a relu activation function. This branch can help to enhance the translation invariance of the features and to learn features from the pooled regions.
- Fifth, it concatenates the outputs of the four branches along the channel dimension to form the output of the inception module. This

output will have the same height and width as the input, but a larger number of channels. The output will contain features from different scales and types of filters, which can enrich the representation of the .input

Flatten

A flatten layer in CNN is a type of layer that reshapes the input feature map into a one-dimensional vector. The flatten layer is used to convert the multidimensional feature maps produced by the convolutional and pooling layers into a single vector that can be fed into a fully connected layer or a dense layer. The flatten layer does not change the number of elements in the input, but only changes their shape. For example, if the input feature map has a shape of (8, 8, 64), the flatten layer will .(,reshape it into a vector of shape (4096

The flatten layer is often used as a transition layer between the convolutional and pooling layers and the fully connected layers in a CNN. The fully connected layers are the final layers of a CNN that perform a linear transformation on the input vector, followed by an activation function, to produce the output vector. The output vector can represent the class probabilities, the regression values, or any other desired output of the CNN. The flatten layer is necessary to bridge the gap between the spatially structured feature maps and the linearly structured output vector. The flatten layer can also help to reduce the number of parameters and the computational cost of the model, as well as to prevent overfitting. However, the flatten layer can also lose some spatial information and correlation between the features, which can affect the performance of the model. Therefore, some CNN architectures, such as ResNet and DenseNet, use global average pooling

or global max pooling layers instead of flatten layers to preserve some spatial information and reduce the dimensionality of the feature maps.

Dence

A dense layer in CNN is a type of layer that performs a linear transformation on the input vector, followed by an activation function. A dense layer is also known as a fully connected layer, because each unit in the layer is connected to every unit in the previous layer. A dense layer can learn the weights and biases that map the input vector to the output vector. A dense layer can be used to perform classification, regression, or any other desired task on the input data.

The code output_layer = Dense(10, activation='softmax')(layer) creates a dense layer that takes the previous layer as the input and produces an output vector of size 10 with a softmax activation function. The softmax activation function is a type of function that normalizes the output vector into a probability distribution over the 10 classes. The softmax function ensures that the sum of the output values is 1, and that each value is between 0 and 1. The softmax function can help the model to output the most likely class for each input image. The output layer is the final layer of the CNN that produces the prediction of the model.

Dropout

Batch normalization

Dropout and batch normalization are two techniques that can help to improve the performance and generalization of a convolutional neural network.

Dropout is a technique that randomly drops out some units in a layer during the training process. This means that some units will not receive any input or produce any output for a given training batch. The dropout rate is a hyperparameter that

controls the probability of dropping out a unit. Dropout can help to reduce the overfitting of the model, as it prevents the co-adaptation of features and introduces some noise and regularization in the network. Dropout can also help to increase the diversity and robustness of the features learned by the model.

Batch normalization is a technique that normalizes the input or output of a layer by subtracting the mean and dividing by the standard deviation of the batch. This means that the input or output of a layer will have zero mean and unit variance for each batch. Batch normalization can help to accelerate the training process, as it reduces the internal covariate shift and allows for higher learning rates. Batch normalization can also help to stabilize the gradient flow and reduce the dependency on the initialization of the weights.

The code layer = Dropout(0.5)(layer) creates a dropout layer that takes the previous layer as the input and applies a dropout rate of 0.5 to it. This means that half of the units in the previous layer will be randomly dropped out for each training batch. The code layer = BatchNormalization()(layer) creates a batch normalization layer that takes the previous layer as the input and normalizes it by subtracting the mean and dividing by the standard deviation of the batch. This means that the output of the previous layer will have zero mean and unit variance for each batch.

۴.

همانگونه که دیدیم بدون دراپ اوت و بچ نرمالزیشن هم به دقت بالای ۸۰ رسیدیم اما با دراپ اوت و بچ نرمالیزیشن قطعا می توانیم در ایپوک ها با بچ سایز های پایین تری به این مقدار رسید که این را هم پیاده سازی کردیم با پارامتر ها و هزینه محاسباتی کمتر و اموزش دادن کم تر مدل با این دو روش دقت را در اپیاک های کم تری به دقت بالا رساندیم

```
input_layer = Input(shape=(32, 32, 3))

layer = Conv2D(32, (3,3), padding='same', activation='relu')(input_layer)
layer = Conv2D(64, (3,3), padding='same', activation='relu')(layer)
layer = Conv2D(32, (3,3), padding='same', activation='relu')(layer)
layer= MaxPooling2D(pool_size=(2, 2))(layer)
layer= inception_module(layer)
layer = Conv2D(128, (3,3), padding='same', activation='relu')(layer)
layer = BatchNormalization()(layer)
layer = Dropout(0.5)(layer)
layer = Conv2D(256, (3,3), padding='same', activation='relu')(layer)
layer = BatchNormalization()(layer)
layer = Dropout(0.5)(layer)
layer = Flatten()(layer)
output_layer = Dense(10, activation='softmax')(layer)

model = Model(inputs=input_layer, outputs=output_layer)
```

مقدار دقت نهایی :

یک روش منظمسازی (regularization) است که باعث میشود شبکه عصبی از بیشبرازش (overfitting) جلوگیری کند. در این روش، در هر مرحله آموزش، بعضی از نورونهای شبکه را به صورت تصادفی صفر میکنیم و از محاسبات حذف میکنیم. این کار باعث میشود که شبکه عصبی به همه نورونها وابسته نشود و ویژگیهای مهمتر را یاد بگیرد. همچنین این روش شبیه به این است که چندین شبکه عصبی با ساختارهای مختلف را به صورت موازی آموزش دهیم و پیشبینیهای آنها را میانگین بگیریم. این روش به شبکه عصبی اجازه میدهد که ویژگیهای مختلف را در مقیاسهای مختلف استخراج کند و هزینه محاسباتی شبکه را کاهش دهد.

بچ نرمالایزیشن یک روش است که برای آموزش شبکههای عصبی عمیق بسیار مفید است. این روش باعث میشود که ورودیهای یک لایه در هر دسته (mini-batch) نرمال شوند. به این ترتیب، فرآیند یادگیری پایدارتر و سریعتر میشود و نیاز به تعداد کمتری از دورههای آموزش (epoch) کاهش مییابد.

بچ نرمالایزیشن بر اساس این ایده کار میکند که توزیع ورودیهای یک لایه ممکن است در طول آموزش تغییر

کند. این تغییر میتواند باعث شود که شبکه عصبی دائماً هدف متحرکی را دنبال کند و یادگیری را دشوارتر کند. این پدیده را انحراف همبستگی داخلی (internal covariate shift) میگویند. بچ نرمالایزیشن با اعمال نرمالایزیشن به صورت مستقل به هر دسته از ورودیها، این پدیده را کاهش میدهد و باعث میشود که توزیع ورودیها به طور متوسط ثابت بماند.

بچ نرمالایزیشن با استفاده از فرمول زیر ورودیها را نرمال میکند:

$$\hat{x} = \frac{x - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$

که در آن x ورودی اصلی، μ_B میانگین دسته، σ_B^2 واریانس دسته و ϵ یک عدد کوچک برای جلوگیری از تقسیم بر صفر است. این فرمول باعث میشود که ورودیها دارای میانگین صفر و واریانس یک شوند.

بچ نرمالایزیشن همچنین دو پارامتر قابل یادگیری به شبکه عصبی اضافه میکند: یک پارامتر مقیاس (scale) به نماد γ و یک پارامتر انحراف (shift) به نماد β. این دو پارامتر به شبکه عصبی اجازه میدهند که توزیع ورودیها را به صورت دلخواه تغییر دهد. برای مثال، اگر تابع فعالسازی لایه بعدی tanh باشد، ممکن است شبکه عصبی بخواهد که توزیع ورودیها دارای میانگین منفی و واریانس بزرگ باشد. این دو پارامتر با استفاده از فرمول زیر ورودیها را تغییر میدهند:

$$y = \gamma \hat{x} + \beta$$

که در آن y ورودی نهایی است.

بچ نرمالایزیشن دارای چندین مزیت است. اول اینکه باعث میشود که شبکه عصبی سریعتر یاد بگیرد و نیاز به تعداد کمتری از دورههای آموزش داشته باشد. دوم اینکه باعث میشود که شبکه عصبی کمتر به مقدار اولیه وزنها و نرخ یادگیری حساس باشد و بهتر بهینه شود. سوم اینکه باعث میشود که شبکه عصبی دارای منظمسازی (regularization) شود و از بیشبرازش جلوگیری کند. چهارم اینکه باعث میشود که شبکه عصبی بتواند از توابع فعالسازی غیرخطی مانند sigmoid و tanh استفاده کند بدون اینکه دچار مشکلاتی مانند میرایی گرادیان (vanishing gradient) شود.

برای اطلاعات بیشتر میتوانید به این مقاله https://www.baeldung.com/cs/batch-normalization-cnn یا این مقاله https://machinelearningmastery.com/batch-normalization-for-training-of-deep-neural-networks/ مراجعه کنید. همچنین میتوانید چند مثال از استفاده از بچ نرمالایزیشن در شبکههای عصبی را در این لینک https://medium.com/mlearning-ai/batch-normalization-and-its-advantages-a1dc52af83d0 ببینید.

```
# define architecture our net custom
input_layer = Input(shape=(32, 32, 3))
# This code creates a convolutional layer that applies 32 filters of size
3x3 to the input layer. The padding parameter is set to 'same', which
means that the output layer will have the same spatial dimensions as the
input layer. The activation parameter is set to 'relu', which means that
the rectified linear unit function will be applied to the output of the
convolution.
layer = Conv2D(32, (3,3), padding='same', activation='relu')(input_layer)
# This code creates a convolutional layer that applies 32 filters of size
3x3 to the input layer. The padding parameter is set to 'same', which
means that the output layer will have the same spatial dimensions as the
input layer. The activation parameter is set to 'relu', which means that
the rectified linear unit function will be applied to the output of the
convolution.
layer = Conv2D(64, (3,3), padding='same', activation='relu')(layer)
layer = Conv2D(32, (3,3), padding='same', activation='relu')(layer)
# This code performs the following operations on the layer:

# •  First, it applies a max pooling layer that reduces the spatial
dimensions of the layer by half. The pool_size parameter is set to (2, 2),
which means that the layer will be divided into 2x2 non-overlapping
regions, and the maximum value in each region will be selected as the
output. This can help to reduce the number of parameters and the
computational cost of the model, as well as to enhance the translation
invariance of the features.

# •  Second, it applies an inception module that combines multiple
convolutional filters of different sizes and a max pooling layer on the
layer. The inception module is a network structure that can increase the
efficiency and performance of the model by allowing it to learn features
at multiple scales and levels of abstraction. The inception module
consists of four parallel branches: a 1x1 convolution, a 1x1 convolution
followed by a 3x3 convolution, a 1x1 convolution followed by a 5x5
convolution, and a 3x3 max pooling followed by a 1x1 convolution. The
```

```
outputs of these branches are then concatenated along the channel
dimension to form the output of the inception module.

# •  Third, it applies another convolutional layer that applies 128
filters of size 3x3 to the output of the inception module. The padding
parameter is set to 'same', which means that the output layer will have
the same spatial dimensions as the input layer. The activation parameter
is set to 'relu', which means that the rectified linear unit function will
be applied to the output of the convolution. This layer can further
extract more features from the inception module output.
layer= MaxPooling2D(pool_size=(2, 2))(layer)
layer= inception_module(layer)
layer = Conv2D(128, (3,3), padding='same', activation='relu')(layer)
  # Batch Normalization
layer = BatchNormalization()(layer)
 #Dropout with 0.5 %
layer = Dropout(0.5)(layer)
layer = Conv2D(256, (3,3), padding='same', activation='relu')(layer)
  # Batch Normalization
layer = BatchNormalization()(layer)
  #Dropout with 0.5 %
layer = Dropout(0.5)(layer)
# make flat


# •  First, it applies a flatten layer that reshapes the layer into a one-
dimensional vector. The flatten layer is used to convert the
multidimensional feature maps into a single vector that can be fed into a
fully connected layer. The flatten layer does not change the number of
elements in the layer, but only changes their shape.

# •  Second, it applies a dense layer that creates a fully connected layer
with 10 units and a softmax activation function. The dense layer is a type
of layer that performs a linear transformation on the input vector,
followed by an activation function. The dense layer can learn the weights
and biases that map the input vector to the output vector. The softmax
activation function is a type of function that normalizes the output
vector into a probability distribution over the 10 classes. The softmax
function ensures that the sum of the output values is 1, and that each
value is between 0 and 1. The softmax function can help the model to
output the most likely class for each input image.
layer = Flatten()(layer)
output_layer = Dense(10, activation='softmax')(layer)
# creat model
model = Model(inputs=input_layer, outputs=output_layer)
```

```
Epoch 1/10
782/782 [==============================] - 1500s 2s/step - loss: 1.3668 - accuracy: 0.5086 - val_loss: 1.0568 - val_accuracy: 0.6281
Epoch 2/10
782/782 [==============================] - 1468s 2s/step - loss: 0.9428 - accuracy: 0.6701 - val_loss: 0.8896 - val_accuracy: 0.6865
Epoch 3/10
782/782 [==============================] - 1563s 2s/step - loss: 0.7603 - accuracy: 0.7376 - val_loss: 0.7905 - val_accuracy: 0.7257
Epoch 4/10
782/782 [==============================] - 1481s 2s/step - loss: 0.6302 - accuracy: 0.7822 - val_loss: 0.7435 - val_accuracy: 0.7407
Epoch 5/10
782/782 [==============================] - 1468s 2s/step - loss: 0.5209 - accuracy: 0.8182 - val_loss: 0.7669 - val_accuracy: 0.7568
Epoch 6/10
782/782 [==============================] - 1462s 2s/step - loss: 0.4172 - accuracy: 0.8533 - val_loss: 0.8180 - val_accuracy: 0.7465
Epoch 7/10
782/782 [==============================] - 1465s 2s/step - loss: 0.3256 - accuracy: 0.8856 - val_loss: 0.8400 - val_accuracy: 0.7526
Epoch 8/10
782/782 [==============================] - 1461s 2s/step - loss: 0.2499 - accuracy: 0.9113 - val_loss: 0.9276 - val_accuracy: 0.7411
Epoch 9/10
 27/782 [>.............................] - ETA: 22:11 - loss: 0.1606 - accuracy: 0.9439
```

بدلیل اینکه مدل خیلی ترین زمان بری داشت همینجا متوقف کردم اما در دو ایپاک دیگر درصد دقت ان به بالای ۹۶ می رسد