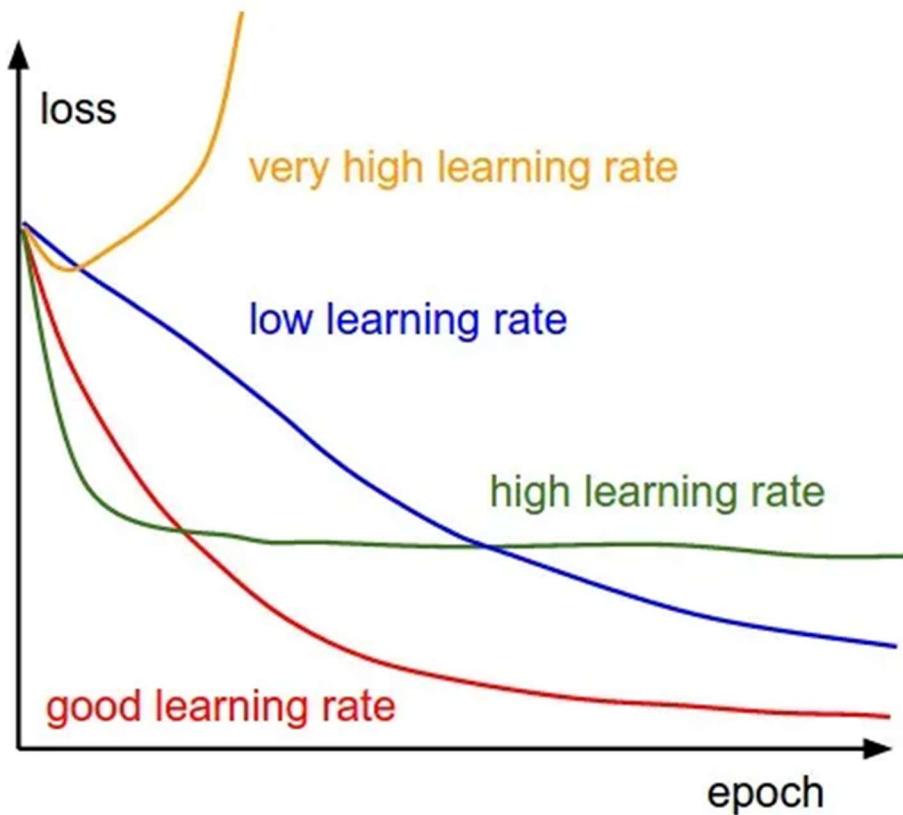


به نام خدا

تمرین سری سوم
درس مبانی یادگیری عمیق
دکتر مرضیه داود آبادی

فرزان رحمانی
۹۹۵۲۱۲۷۱

سوال اول



الف) استفاده از نرخ یادگیری بسیار بالا می تواند باعث شود که الگوریتم بهینه سازی از نقطه بهینه در فضای پارامتر فراتر رفته و منجر به واگرایی شود. به جای همگرایی به مینیمم، ضرر مدل ممکن است افزایش یابد یا به شدت نوسان کند.

آموختن ناپایدار (Unstable Training): فرآیند آموختن ممکن است ناپایدار شود و همگرایی مدل به یک جواب بهینه را دشوار کند.

همچنین همان طور که در نمودار بالا دیده می شود اگر نرخ آموزش خیلی زیاد(very high learning rate) باشد ضرر با افزایش epoch ها بیشتر می شود و مدل بجای همگرا شدن واگرا می شود. همچنین اگر نرخ آموزش زیاد باشد(high learning rate) ضرر از یک حدی به بعد دیگر بهتر نمی شود با ادامه دادن epoch در جواب نا بهینه ثابت می ماند.

چگونه می توان این مشکلات را تشخیص داد:

ضرر انفجاری(Exploding Loss): یک نشانه واضح از نرخ بالای یادگیری، افزایش سریع تابع ضرر در طول آموزش است. عملکرد نامنظم مدل(Erratic Model Performance): مشاهده نوسانات شدید یا بی ثباتی در معیارهای عملکرد مدل (مانند دقت یا ضرر) دوره به دوره(epoch by epoch) است.

Monitoring training loss: اگر ضرر تمرين در حال افزایش یا نوسان باشد، ممکن است نشان دهنده این باشد که میزان یادگیری بسیار بالاست.

Checking weight updates: اگر به روز رسانی وزن بسیار زیاد باشد، نشانه آن است که میزان یادگیری بسیار زیاد است. ارزیابی عملکرد مدل بر روی داده های اعتبارسنجی: اگر عملکرد مدل بر روی داده های اعتبارسنجی بهبود نمی یابد یا حتی در حال کاهش است، ممکن است به دلیل نرخ یادگیری بالا باشد.

(ب)

همگرایی آهسته: استفاده از نرخ یادگیری بسیار پایین می تواند منجر به همگرایی آهسته شود و باعث می شود فرآیند آموزش زمان زیادی را برای رسیدن به یک راه حل بهینه صرف کند. در نتیجه هزینه های آموزش مانند استفاده از GPU افزایش می باید.

گیر افتادن در مینیمم ها یا فلات های محلی(Getting Stuck in Local Minima or Plateaus): نرخ یادگیری بسیار پایین ممکن است مانع از فرار مدل از مینیمم ها یا فلات های محلی شود و مانع از توانایی آن برای یافتن راه حل های بهتر شود.

همچنین همان طور که در نمودار بالا می بینیم نرخ آموزش پایین با گذشت دوره(epoch) ها تغییر کاهش کمتری در ضرر ایجاد می کند که با توجه به اینکه در نهایت به جواب همگرایی رسد ولی خیلی طور می کشد و نیاز به زمان بسیار زیادی دارد.

تشخیص:

راکد شدن یا کاهش آهسته ضرر: اگر تابع زیان بسیار آهسته کاهش می یابد یا به نظر می رسد در یک مقدار بزرگ گیر کرده است بدون اینکه در طول دوره ها پیشرفت زیادی داشته باشد.

عملکرد ضعیف مدل: علیرغم آموزش برای مدت زمان قابل توجهی، معیارهای عملکرد مدل پایین می مانند یا بهبود نمی یابند. ارزیابی عملکرد مدل بر روی داده های اعتبارسنجی: اگر عملکرد مدل بر روی داده های اعتبارسنجی بهبود نمی یابد یا بسیار کم بهبود می یابد، ممکن است به دلیل نرخ یادگیری پایین باشد.

چگونه می توان مشکلات نرخ آموزش کم یا زیاد را حل کنیم؟

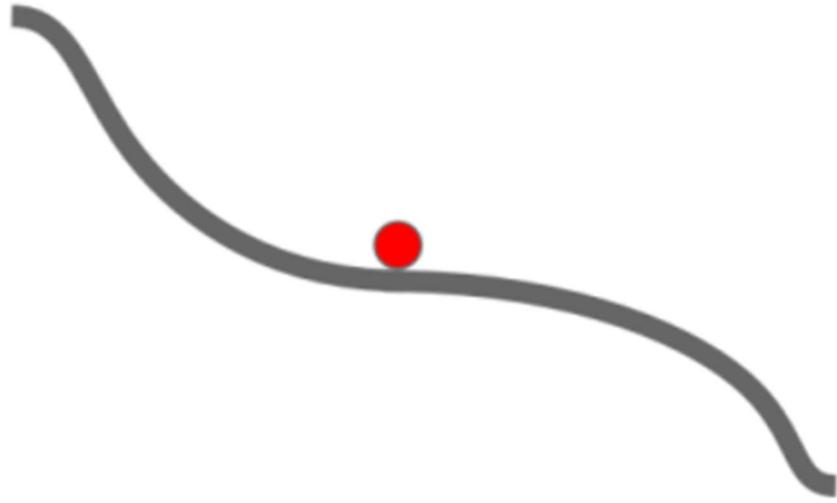
۱. Learning Rate Schedulers: نرخ یادگیری را به طور پویا در طول آموزش با استفاده از زمانبندهای نرخ یادگیری، مانند کاهش تدریجی نرخ یادگیری (مثلاً با استفاده از کاهش نرخ یادگیری) یا استفاده از روش های نرخ یادگیری تطبیقی (مانند RMSprop، Adam).

۲. منحنی های یادگیری را رصد کنید: ضرر آموزش و اعتبارسنجی یا سایر معیارهای ارزیابی در برابر دوره ها را ترسیم کنید. این به شناسایی مسائلی مانند افزایش سریع یا رکود در عملکرد کمک می کند.

۳. تنظیم فرایپارامتر: با نرخ های یادگیری مختلف آزمایش کنید و تأثیرات آنها را بر همگرایی مدل و عملکرد روی یک مجموعه داده اعتبارسنجی مشاهده کنید. جستجوی شبکه ای یا جستجوی تصادفی می تواند برای بهینه سازی های پردازش استفاده شود.

۴. توقف زودهنگام: در صورت عدم بهبود قابل توجهی در عملکرد اعتبارسنجی در طول تعداد معینی از دوره ها، توقف زودهنگام را اجرا کنید تا آموزش را متوقف کنید.

(ب)



همان طور که در شکل بالا می بینید نقطه زینی یک نقطه بحرانی در فضای بهینه سازی تابع هدف است که در آن گرادیان صفر است (مانند مینیمم) اما مینیمم نیست. در عوض، این نقطه ای است که در آن تابع از یک طرف افزایش می باید و از طرف دیگر کاهش می باید، که شبیه به شکل یک زین است. این بدان معنی است که مقدار تابع می تواند در جهات مختلف از نقطه زین افزایش یا کاهش یابد. در یک نقطه زینی، گرادیان ممکن است در همه ابعاد ناپدید شود، و برای الگوریتم های بهینه سازی تمايز بین نقاط زین و مینیمم واقعی دشوار باشد. در واقع، نقاط زینی می توانند برای الگوریتم های بهینه سازی چالشی ایجاد کنند، زیرا می توانند مدل را در یک راه حل غیربهینه به دام بیندازند. الگوریتم های مختلف توانایی های متفاوتی برای مقابله با نقاط زین دارند.

Adam در مقابل SGD در برخورد با نقاط زین:

نزول گرادیان تصادفی (SGD):

مزایا:

садگی: اجرای SGD نسبتاً ساده است و از نظر محاسباتی در مقایسه با الگوریتم های پیچیده تر ساده تر است و پیچیدگی کمتری دارد. تعمیم: در سناریوهای خاص، SGD به دلیل ماهیت تصادفی ذاتی خود می تواند بهتر تعمیم دهد، که می تواند به فرار از نقاط زینی کمک کند.

ضمین همگرایی در مسائل بهینه سازی محدب: SGD دارای ضمانت های همگرایی نظری برای مسائل بهینه سازی محدب است.

معایب:

سرعت همگرایی: SGD می تواند برای همگرایی موثر، به ویژه در نزدیکی نقاط زین یا در مناطقی با انحنای بالا، مشکل داشته باشد. حساسیت به نرخ یادگیری: همگرایی آن به شدت به نرخ یادگیری به خوبی تنظیم شده بستگی دارد، و آن را مستعد گیر کردن در نقاط زین یا نوسان در اطراف آنها می کند. در واقع نرخ یادگیری بالا می تواند منجر به نوسان یا واگرایی شود، در حالی که نرخ یادگیری پایین می تواند منجر به همگرایی کند یا گیر کردن در نقاط زینی شود.

Adam (تخمین لحظه تطبیقی):

مزایا:

نرخ های یادگیری تطبیقی: Adam از نرخ های یادگیری تطبیقی برای هر پارامتر استفاده می کند و به آن اجازه می دهد تا مقادیر مختلف گرادیان ها را به طور موثر مدیریت کند. در واقع Adam به طور خودکار نرخ یادگیری را برای هر پارامتر بر اساس تاریخچه گرادیان ها تنظیم می کند و آن را کمتر مستعد گیر کردن در نقاط زین می کند.

Momentum: برای افزایش سرعت همگرایی، به ویژه در حضور شبکه های پراکنده، momentum را در خود جای می دهد. در حقیقت، Adam از Momentum استفاده می کند که به مدل کمک می کند با تجمع سرعت در جهت گرادیان، از نقاط زینی فرار کند و نرخ های یادگیری تطبیقی که به همگرایی سریع تر مدل کمک می کند.

معايير:

پیچیدگی: Adam پیچیده‌تر از SGD است، که شامل محاسبات و فرآپارامترهای اضافی (مانند momentum decay rates and bias correction) است که نیاز به تنظیم دارند.

حافظه و محاسبات: نیاز به ذخیره و به روز رسانی میانگین متحرک نمایی گرادیان‌ها و مربع‌های آنها برای هر پارامتر دارد که می‌تواند حافظه و محاسبات بیشتری را مصرف کند.

برخورد با نقاط زینی:

SGD به دلیل اینکه به نخ یادگیری ثابت، در نقاط زینی بیشتر با مشکل موواجه می‌شود. ممکن است گیر کند یا به آرامی در اطراف این مناطق همگرا شود.

Adam اغلب در اطراف نقاط زین به دلیل نخ یادگیری تطبیقی و Momentum عمل می‌کند و به آن امکان می‌دهد از نقاط زینی به طور موثرتر از SGD فرار کند.

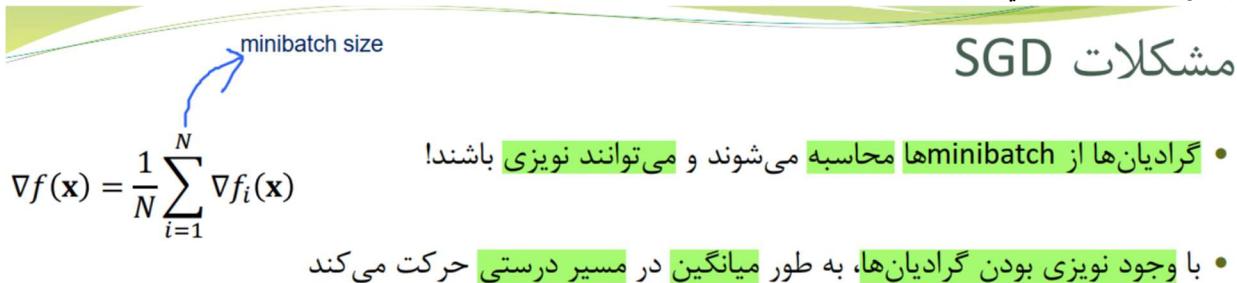
خلاصه:

SGD ساده است اما ممکن است به آرامی همگرا شود یا در اطراف نقاط زین بدون تنظیم مناسب نخ یادگیری گیر کند. Adam پیچیده‌تر است، اما اغلب سریع‌تر همگرا می‌شود و به دلیل Momentum و حرکت تطبیقی‌اش، در پیمایش نقاط زینی ماهرتر است و بهتر از SGD عمل می‌کند.

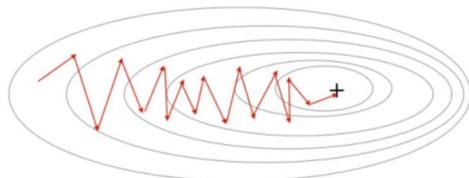
با این حال، اثربخشی این الگوریتم‌ها به مسئله خاص، مجموعه داده‌ها و تنظیمات فرآپارامتر بستگی دارد. گاهی اوقات، اصلاحات ساده در این الگوریتم‌ها یا استفاده از تکنیک‌ها (مانند استفاده از زمان‌بندی نخ یادگیری با SGD یا استفاده از بهینه‌سازهای مختلف در ترکیب) می‌تواند منجر به عملکرد بهتر در برخورد با نقاط زینی و بهینه‌سازی به طور کلی شود. یا مثلاً SGD ساده تر و از نظر محاسباتی کارآمدتر است و دارای ضمانت‌های همگرایی تئوری برای مسائل بهینه‌سازی محدب است.

(ت)

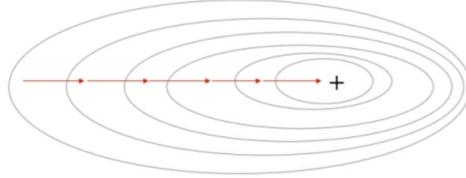
این سوال شبیه به مثال زیر در اسلاید‌ها است با این تفاوت که GD mini-batch عملکردی ما بين GD و SGD دارد.



Stochastic Gradient Descent



(Batch) Gradient Descent



SGD , mini-batch GD, batch GD(GD)

۱۷

بر اساس جزئیات تصویری که ارائه کشده است، به نظر می‌رسد که نمودار سمت چپ، احتمالاً نشان‌دهنده شیب نزولی دسته‌ای (batch gradient decent) است، در حالی که نمودار سمت راست نشان‌دهنده شیب نزولی دسته‌ای کوچک (mini-batch gradient decent) است.

در شبیب نزولی دسته‌ای (batch gradient decent)، هزینه به آرامی و نرم (smoothly) با تکرارها کاهش می‌یابد زیرا گرادیان تابع هزینه را با توجه به پ‌کل مجموعه داده آموزشی محاسبه می‌کند. در مقابل، شبیب نزولی دسته‌ای کوچک (mini-batch gradient decent) وزن‌ها را بیشتر به روزرسانی (در یک بار دیدن کل مجموعه داده) می‌کند و معمولاً نوسانات بیشتری در کاهش هزینه به دلیل گرادیان‌های محاسباتی در دسته‌های کوچک داده در هر مرحله نشان می‌دهد.

توضیحات بیشتر راجه به دو الگوریتم بهینه ساز:

Batch Gradient Descent (BGD):

BGD یک الگوریتم بهینه سازی ساده و سراسرت است. در هر تکرار، گرادیان تابع ضرر را در کل مجموعه آموزشی محاسبه می‌کند و سپس پارامترهای مدل را در جهت مخالف گرادیان به روز می‌کند.

- در شبیب نزولی دسته‌ای، الگوریتم گرادیان‌ها را با استفاده از کل مجموعه داده در هر تکرار محاسبه می‌کند.
- نمودار کاهش هزینه برای BGD معمولاً کاهش هموارتر و ثابت در تابع هزینه با افزایش تکرارها را نشان می‌دهد.
- این روش معمولاً به روشی مستقیم تر به سمت حداقل همگرا می‌شود زیرا کل مجموعه داده را در هر تکرار در نظر می‌گیرد.
- نیاز به بارگذاری کل مجموعه آموزشی در حافظه دارد.

Mini-Batch Gradient Descent (MBGD):

MBGD گونه‌ای از BGD است که پارامترهای مدل را در دسته‌های کوچک‌تری از نمونه‌های آموزشی در هر تکرار به روزرسانی می‌کند. این باعث می‌شود MBGD سریعتر از BGD باشد، به خصوص برای مجموعه‌های آموزشی بزرگ. با این حال، MBGD می‌تواند نویز بیشتری نسبت به BGD داشته باشد، زیرا گرادیان محاسبه شده روی یک دسته کوچک‌تر از نمونه‌ها ممکن است به عنوان نماینده گرادیان در کل مجموعه آموزشی نباشد.

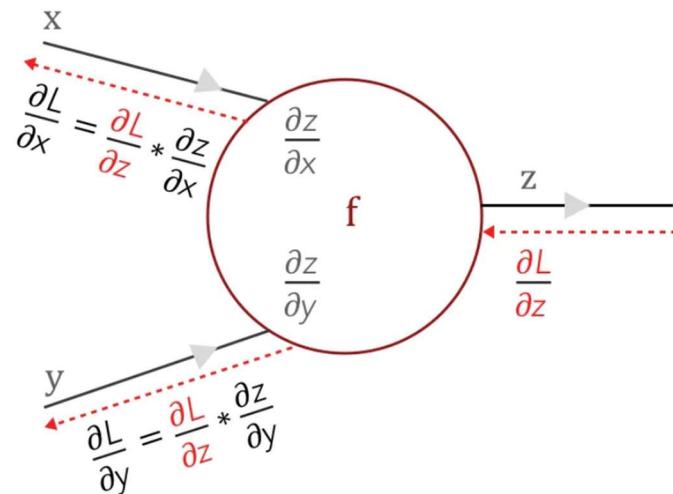
- شبیب نزولی دسته‌ای کوچک، گرادیان‌ها را با استفاده از زیرمجموعه‌ها یا مینی دسته‌های مجموعه داده در هر تکرار محاسبه می‌کند.
- نمودار کاهش هزینه برای نزول MBGD ممکن است در مقایسه با BGD نویزی تر به نظر برسد یا دارای نوسانات بیشتری باشد.
- با توجه به استفاده از زیرمجموعه‌های کوچک‌تر داده‌ها، کاهش هزینه ممکن است بی‌نظمی‌ها یا نوسانات بیشتری را با پیشرفت الگوریتم از طریق تکرار نشان دهد.
- نیازی به بارگذاری کل مجموعه آموزشی در حافظه ندارد.

مراجع:

<https://chat.openai.com/>
<https://bard.google.com/>
<https://claude.ai/chats>

سوال دوم

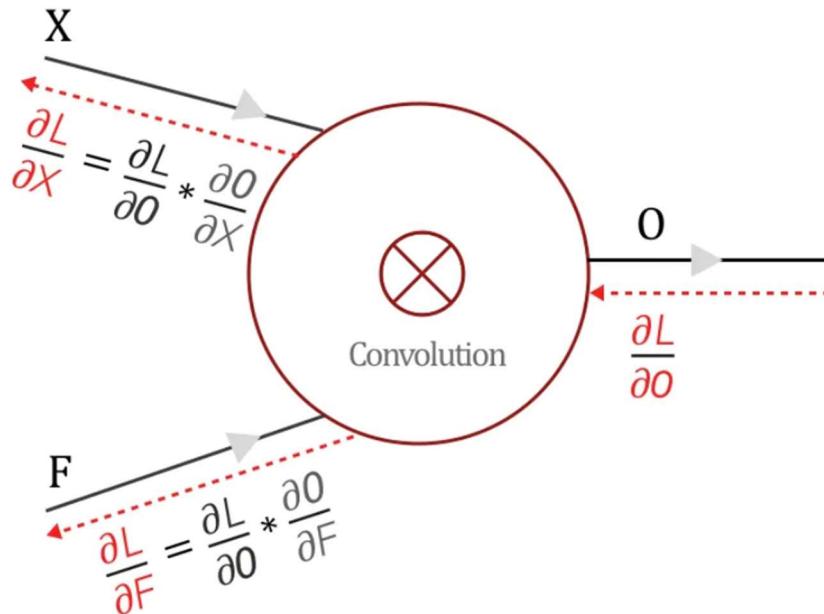
برخی از مهم ترین نکات لینک داده شده که در حل سوال از آنها استفاده کردیم:



$\frac{\partial z}{\partial x}$ & $\frac{\partial z}{\partial y}$ are local gradients

$\frac{\partial L}{\partial z}$ is the loss from the previous layer which has to be backpropagated to other layers

Finding the loss gradients for x and y



Both the Forward pass and the Backpropagation of a Convolutional layer are Convolutions

You highlighted

Summing it up:

Backpropagation in a Convolutional Layer of a CNN

Finding the gradients:

$$\frac{\partial L}{\partial F} = \text{Convolution} \left(\text{Input } X, \text{ Loss gradient } \frac{\partial L}{\partial O} \right)$$

$$\frac{\partial L}{\partial X} = \text{Full Convolution} \left(\text{180}^\circ \text{ rotated Filter } F, \text{ Loss Gradient } \frac{\partial L}{\partial O} \right)$$

How to calculate $\partial L / \partial X$ and $\partial L / \partial F$



کانولوشن و همبستگی

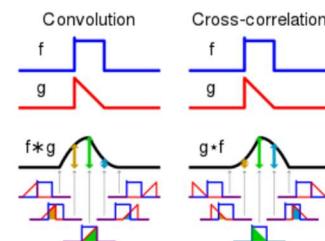
- بسیاری از کتابخانه‌های ML همبستگی متقابل را پیاده‌سازی می‌کنند اما آن را کانولوشن می‌نامند!
- الگوریتم یادگیری مقادیر مناسب هسته را در مکان مناسب یاد می‌گیرد

$$S(i) = (I * K)(i) = \sum_m I(i - m)K(m)$$

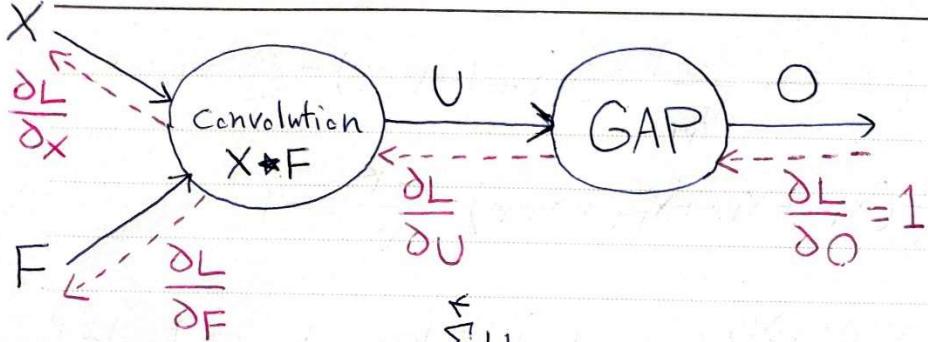
$$S(i) = (I * K)(i) = \sum_m I(i + m)K(m)$$

$$S(i, j) = (I * K)(i, j) = \sum_m \sum_n I(i - m, j - n)K(m, n)$$

$$S(i, j) = (I * K)(i, j) = \sum_m \sum_n I(i + m, j + n)K(m, n)$$



جواب سوال در زیر آمده است:



$$U = X * F, \quad O = \frac{\sum_{j=1}^k U_j}{k} = \frac{U_{11} + U_{12} + U_{21} + U_{22}}{k}, \quad \frac{\partial L}{\partial O} = 1$$

$$\frac{\partial L}{\partial U_i} = \frac{\partial L}{\partial O} \times \frac{\partial O}{\partial U_i} = 1 \times \frac{1}{k} = \frac{1}{k}, \quad \frac{\partial L}{\partial F} = X * \frac{\partial L}{\partial U}$$

$\frac{\partial L}{\partial X} = \text{full convolution} \left(\begin{matrix} 180^\circ \text{ rotated} \\ \text{Filter } F \end{matrix}, \frac{\partial L}{\partial U} \right)$

forward pass

$$U = X * F = \begin{array}{|c|c|} \hline 1 & \lambda \\ \hline -1 & \lambda \\ \hline \end{array} \quad U_{11} = 1 - 1 + 1 = 1, \quad U_{12} = 1 - \lambda - \lambda = -2\lambda$$

$$U_{21} = -1 - 1 - 1 = -3, \quad U_{22} = \lambda + \lambda = 2\lambda$$

$$O = \text{GAP}(U) = \frac{1 + \lambda - 1 - \lambda}{4} = \frac{1}{4}$$

backward pass

$$\frac{\partial L}{\partial O} = 1 \quad \frac{\partial O}{\partial U_i} = \frac{1}{4}$$

$$\frac{\partial L}{\partial F_{11}} = \frac{\partial L}{\partial O} \times \left(\sum_j \frac{\partial O}{\partial U_j} \cdot \frac{\partial U_j}{\partial F_{11}} \right) = 1 \times \left(\frac{1}{4} X_{11} + \frac{1}{4} X_{12} + \frac{1}{4} X_{21} + \frac{1}{4} X_{22} \right) = \frac{1}{4} (1 - 2\lambda - 3 + 2\lambda) = \frac{-1}{4}$$

$$\frac{\partial L}{\partial F_{12}} = \frac{\partial L}{\partial O} \left(\sum_j \frac{\partial O}{\partial U_j} \cdot \frac{\partial U_j}{\partial F_{12}} \right) = 1 \times \left(\frac{X_{12}}{4} + \frac{X_{21}}{4} + \frac{X_{22}}{4} \right) = \frac{V}{4} = 1, V \lambda$$

Subject
Date

backward pass

$$\frac{\partial L}{\partial F_{11}} = \frac{\partial L}{\partial O} \cdot \left(\sum_j \frac{\partial O}{\partial U_j} \cdot \frac{\partial U_j}{\partial F_{11}} \right) = \frac{1}{4}(Y + 1 + F - Y) = \frac{\Delta}{4} = 1, \text{V}\omega$$

$$\frac{\partial L}{\partial Y_2} = 1 \times \frac{1}{4}(X_{22} + X_{23} + X_{32} + X_{33}) = \frac{-F}{4} = -1$$

$$\frac{\partial L}{\partial X_{11}} = \frac{\partial L}{\partial O} \left(\sum_j \frac{\partial O}{\partial U_j} \cdot \frac{\partial U_j}{\partial X_{11}} \right) = 1 \times \left(\frac{1}{4} \times F_{11} + \frac{1}{4} \times 0 + \frac{1}{4} \times 0 + \frac{1}{4} \times 0 \right) = \frac{F}{4} = 0, \omega$$

$$\frac{\partial L}{\partial X_{12}} = \frac{\partial L}{\partial O} \left(\sum_j \frac{\partial O}{\partial U_j} \cdot \frac{\partial U_j}{\partial X_{12}} \right) = 1 \times \left(\frac{F_{11}}{4} + \frac{F_{12}}{4} + \frac{0}{4} + \frac{0}{4} \right) = \frac{F}{4} = 0, \omega$$

$$\frac{\partial L}{\partial X_{13}} = 1 \times \left(\frac{0}{4} + \frac{F_{12}}{4} + \frac{0}{4} + \frac{0}{4} \right) = 0, \quad \frac{\partial L}{\partial X_{21}} = 1 \times \left(\frac{F_{11}}{4} + \frac{0}{4} + \frac{F_{12}}{4} + \frac{0}{4} \right) = 1, \text{V}\omega$$

$$\frac{\partial L}{\partial X_{22}} = 1 \times \left(\frac{F_{12}}{4} + \frac{F_{21}}{4} + \frac{F_{12}}{4} + \frac{F_{11}}{4} \right) = 0, \quad \frac{\partial L}{\partial X_{23}} = \left(\frac{0}{4} + \frac{F_{22}}{4} + 0 + \frac{F_{12}}{4} \right) = 0, \text{V}\omega$$

~~$$\frac{\partial L}{\partial X_{21}} = \frac{0}{4} + \frac{0}{4} + \frac{F_{21}}{4} + \frac{0}{4} = 0, \text{V}\omega, \quad \frac{\partial L}{\partial X_{33}} = \frac{1}{4}(0 + 0 + F_{22} + F_{11}) = 0, \text{V}\omega$$~~

$$\frac{\partial L}{\partial X_{33}} = \frac{\partial L}{\partial O} \cdot \left(\sum_j \frac{\partial O}{\partial U_j} \cdot \frac{\partial U_j}{\partial X_{33}} \right) = 1 \times \left(\frac{1}{4} \times 0 + \frac{1}{4} \times 0 + \frac{1}{4} \times 0 + \frac{1}{4} \times F_{22} \right) = \frac{1}{4}$$

پس در نهایت گردیان های این لایه هم کشی به شکل زیری شود:

$$\frac{\partial L}{\partial F} = \begin{bmatrix} 2, \omega & 1, \text{V}\omega \\ 1, \text{V}\omega & -1 \end{bmatrix}$$

$$\frac{\partial L}{\partial X} = \begin{bmatrix} 0, \omega & 0, \omega & 0 \\ 1, \text{V}\omega & 0 & 0, \text{V}\omega \\ -0, \text{V}\omega & -0, \omega & 0, \text{V}\omega \end{bmatrix}$$

مراجع:

<https://chat.openai.com/>
<https://bard.google.com/>
<https://claude.ai/chats>
<https://pavij.medium.com/convolutions-and-backpropagations-46026a8f5d2c>

سوال سوم

(الف)

می خواهیم تعداد پارامتر های شبکه معرفی شده در کد زیر را محاسبه کیم:

```
model = Sequential()
model.add(Input (shape=(500, 7)))
model.add(Conv1D (filters=16, kernel_size=3, activation="relu"))
model.add(MaxPool1D())
model.add(Conv1D (filters=32, kernel_size=5, activation="relu"))
model.add (MaxPool1D())
model.add(Conv1D (filters=64, kernel_size=5, activation="relu"))
model.add(MaxPool1D() )
model.add(Flatten())
model.add(Dense (units=128, activation="relu"))
model.add(Dense (units=5, activation="softmax"))
```

برای محاسبه تعداد پارامترهای هر لایه، باید فرمول های محاسبه پارامتر در هر نوع لایه را در نظر بگیریم.

با فرض اینکه:

برای conv : $padding=0$ و $stride=1$

برای pool : $padding=0$ و $stride=2$

1. **Input Layer:** There are no parameters in the input layer because it simply receives the input data.
2. **Conv1D Layer:** Number of parameters for a Conv1D layer can be calculated using the formula:
$$\text{Number of Parameters} = (\text{kernel_size} \times \text{input_channels} + 1) \times \text{filters}$$
3. **MaxPooling1D Layer:** MaxPooling layers do not have any parameters. They perform a fixed operation.
4. **Flatten Layer:** The Flatten layer doesn't have any parameters; it simply flattens the input for the subsequent Dense layer.
5. **Dense Layer:** The number of parameters in a Dense layer is calculated as:
$$\text{Number of Parameters} = (\text{input_units} + 1) \times \text{output_units}$$

حال بباید تعداد پارامترهای هر لایه را در مدل ارائه شده محاسبه کنیم:

1. **Input Layer:** No parameters.
2. **Conv1D Layer 1:**

- Filters = 16
- Kernel Size = 3
- Input channels = 7
- Parameters=($3 \times 7 + 1$) $\times 16 = 352$ parameters

3. **MaxPooling1D Layer 1:** No parameters.

4. **Conv1D Layer 2:**

- Filters = 32
- Kernel Size = 5
- Parameters=($5 \times 16 + 1$) $\times 32 = 2592$ parameters

5. **MaxPooling1D Layer 2:** No parameters.

6. **Conv1D Layer 3:**

- Filters = 64
- Kernel Size = 5
- Parameters=($5 \times 32 + 1$) $\times 64 = 10304$ parameters

7. **MaxPooling1D Layer 3:** No parameters.

8. **Flatten Layer:** No parameters.

9. **Dense Layer 1:**

- Input Shapes from input layer up to this layer: 500,7 { $500 - 3 + 1 = 498$ } $\rightarrow 498 \times 16 \{ (498 - 2) / 2 + 1 = 249 \} \rightarrow 249 \times 16 \{ 249 - 5 + 1 = 245 \} \rightarrow 245 \times 32 \{ (245 - 2) / 2 + 1 = 122 \} \rightarrow 122 \times 32 \{ 122 - 5 + 1 = 118 \} \rightarrow 118 \times 64 \{ (118 - 2) / 2 + 1 = 59 \} \rightarrow 59 \times 64 \rightarrow 3776$
- Input Units = number of units from the previous layer ($59 \times 64 = 3776$)
- Output Units = 128
- Parameters=(($3776 + 1$) $\times 128 = 483456$ parameters)

10. **Dense Layer 2:**

- Input Units = 128
- Output Units = 5
- Parameters=($128 + 1$) $\times 5 = 645$ parameters

بنابراین، در مجموع، تعداد پارامترهای مدل داده شده، مجموع پارامترهای هر لایه خواهد بود:

$$\text{total parameters} = 0 + 352 + 0 + 2592 + 0 + 10304 + 0 + 0 + 483456 + 645 = \\ 352 + 2592 + 10304 + 483456 + 645 = 497349 \text{ parameters}$$

(ب)

هر دو لایه Conv3D و Conv2D انواع لایه های کانولوشنی هستند که در شبکه های عصبی کانولوشنی (CNN) و مدل های یادگیری عمیق برای استخراج ویژگی های فضایی از داده های ورودی استفاده می شوند. با این حال، آنها در ابعاد داده های ورودی که می توانند مدیریت کنند، متفاوت هستند. تفاوت اصلی بین آنها این است که Conv2D برای پردازش داده های دو بعدی مانند تصاویر استفاده می شود، در حالی که Conv3D برای پردازش داده های سه بعدی مانند فیلم ها یا اسکن های پیشکی سه بعدی یا تصاویر ماهواره ای با کanal های زیاد استفاده می شود.

لایه های Conv2D برای پردازش داده های دو بعدی مانند تصاویر طراحی شده اند. آنها یک فیلتر با اندازه از پیش تعريف شده را روی تصویر ورودی می پیچانند تا یک نقشه ویژگی تولید کنند. اندازه نقشه ویژگی خروجی به اندازه فیلتر، گام و لایه (filter, the stride, گام و padding) بستگی دارد.

از طرف دیگر لایه های Conv3D برای پردازش داده های حجمی مانند فیلم ها یا اسکن های پیشکی طراحی شده اند. آنها یک فیلتر سه بعدی را روی داده های ورودی می پیچند تا یک حجم ویژگی چهار بعدی تولید کنند. اندازه حجم ویژگی خروجی بستگی به اندازه فیلتر، گام و لایه دارد.

در Conv2D، هسته به صورت دو بعدی روی تصویر ورودی می لغزد و در هر موقعیت یک ضرب نقطه ای بین هسته (kernel) و ورودی انجام می دهد. این عملیات برای هر کanal از تصویر ورودی تکرار می شود و یک نقشه ویژگی دو بعدی تولید می کند. از طرف دیگر، Conv3D یک هسته سه بعدی را روی حجم ورودی می کشد و در هر موقعیت یک محصول نقطه ای بین هسته و ورودی انجام می دهد. این عملیات برای هر کanal از حجم ورودی تکرار می شود و یک نقشه ویژگی سه بعدی تولید می کند.

کاربردهای لایه های Conv3D:

- تجزیه و تحلیل ویدیو: لایه های Conv3D معمولاً در کارهای تجزیه و تحلیل ویدیو مانند تشخیص عمل (action recognition)، ردیابی حرکت (motion tracking) و تشخیص ناهمجارتی (anomaly detection) استفاده می شود. آنها می توانند ویژگی های مکانی-زمانی را از فریم های ویدیو استخراج کنند و به شبکه اجازه می دهند الگوها و پویایی ویدیو را بیاموزند.
- تصویربرداری پیشکی: لایه های Conv3D همچنین در برنامه های تصویربرداری پیشکی مانند تقسیم بندی تصویر (image segmentation)، تشخیص ضایعه (lesion detection) و طبقه بندی بیماری (disease classification) استفاده می شوند. آنها می توانند اسکن های پیشکی حجمی، مانند CT اسکن یا اسکن MRI را برای شناسایی و طبقه بندی ناهمجارتی ها تجزیه و تحلیل کنند.
- تشخیص اشیاء سه بعدی: لایه های Conv3D را می توان برای کارهای تشخیص اشیاء سه بعدی، مانند تشخیص اشیاء و طبقه بندی (object detection and classification) در داده های ابر نقطه سه بعدی (3D point cloud) استفاده کرد. آنها می توانند ویژگی های فضایی را از ابر نقطه سه بعدی استخراج کنند و به شبکه اجازه می دهند اشیاء را در صحنه تشخیص و طبقه بندی کنند.

به طور کلی، لایه های Conv3D به دلیل افزایش ابعاد داده های ورودی و فیلترها، از نظر محاسباتی گرانتر از لایه های Conv2D هستند. با این حال، با افزایش در دسترس بودن داده های حجمی و افزایش قدرت محاسباتی پردازنده های گرافیک، آنها اهمیت فرازینده ای پیدا می کنند.

جواب سوال به بیان chat gpt

The main difference between Conv2D and Conv3D layers lies in the dimensionality of the input data they process and the spatial understanding they possess.

1. Conv2D (Two-Dimensional Convolutional Layer):

- Processes two-dimensional input data, typically used for images (height x width x channels).
- Uses a 2D kernel/filter that moves across the height and width dimensions of the input.
- Commonly applied in image-related tasks like image classification, object detection, and segmentation.

2. Conv3D (Three-Dimensional Convolutional Layer):

- Processes three-dimensional input data, which includes depth in addition to height and width (volume data - height x width x depth x channels).
- Uses a 3D kernel/filter that moves across the height, width, and depth dimensions of the input.
- Applications include video data, medical imaging (like CT or MRI scans), volumetric images, 3D object recognition, and spatiotemporal data analysis.

Applications of Conv3D layers: The Conv3D layer is particularly useful in handling spatiotemporal data where the temporal dimension (time) is also important. Some applications include:

1. **Video Analysis:** Conv3D layers are valuable for tasks involving video understanding, action recognition, video classification, and video segmentation where both spatial and temporal information is crucial.
2. **Medical Imaging:** For volumetric medical imaging data such as CT scans, MRI sequences, or 3D microscopy data, Conv3D networks are employed to extract spatial patterns across multiple layers (slices) and over time, aiding in diagnosis or analysis.
3. **Robotics and Autonomous Vehicles:** Conv3D networks can be applied in robotics and autonomous vehicles to process data from 3D sensors or multiple cameras to understand the environment in 3D space over time.
4. **Spatiotemporal Analysis:** Tasks involving analysis of spatiotemporal data like weather forecasting, fluid dynamics simulations, and analyzing seismic data can benefit from Conv3D networks to capture both spatial and temporal dependencies within the data.

In essence, Conv3D layers extend the capabilities of convolutional neural networks to handle spatiotemporal information, enabling them to understand and process three-dimensional data with temporal dependencies, which is essential in various real-world applications.

مراجع:

- <https://chat.openai.com/>
- <https://bard.google.com/>
- <https://claude.ai/chats>
- <https://forums.fast.ai/t/what-is-the-difference-between-2d-vs-3d-convolutions/52495>

سوال چهارم

نوت بوک پیوست شده HW4.ipynb با توجه به موارد خواسته شده در آن تکمیل شده است. همان طور که مبینیم تصاویر دادگان تومور مغزی که در یک مسیر مشخص قرار دارند خوانده شده اند. و سپس از آنها برآ آموزش و ارزیابی شبکه استفاده کرده ایم. سپس با استفاده از SequentialAPI و FunctionalAPI دو مدل شبکه عصبی با معماری یکسان تعریف کرده ایم و نتایج تقریباً مشابهی را داده اند.

توضیح مختصر کد پیاده سازی شده:

کد پایتون پیاده سازی شده چندین کار مرتبط با یادگیری عمیق را با استفاده از TensorFlow و Keras برای ساخت، آموزش و ارزیابی مدل های شبکه عصبی کانولوشن (CNN) برای طبقه بندی تصویر انجام می دهد. در اینجا خلاصه ای از کارهایی که هر بخش انجام می دهد آورده شده است:

واردات و راه اندازی(Imports and Setup): کد با وارد کردن کتابخانه های ضروری مانند TensorFlow، Keras و Matplotlib کد از Imports and Setup شروع می شود. همچنین یک مجموعه داده را از یک پیوند Google Drive دانلود می کند، آن را استخراج می کند و دایرکتوری کاری را تنظیم می کند.

بارگذاری مجموعه داده (Load Dataset):

- کد از tf.keras.preprocessing.image_dataset_from_directory برای بارگذاری تصاویر از فهرست مجموعه داده های ارائه شده استفاده می کند.
- این پارامترها مانند برچسب ها، حالت رنگ، اندازه تصویر، اندازه دسته ای و تقسیم اعتبار را تنظیم می کند.

نمایش نمونه های مجموعه داده (Display Dataset Samples): تصاویر نمونه را از مجموعه داده های بارگذاری شده نمایش می دهد تا داده هایی را که برای آموزش استفاده می شود به تصویر بکشد.

ساخت و آموزش مدل Sequential API:

- یک مدل CNN با استفاده از Sequential API (پشته خطی لایه ها) ایجاد می کند.
- لایه ها را برای لایه های کانولوشن، flattening، max-pooling، و لایه های متراکم تعریف می کند.
- مدل را با بهینه ساز، تابع ضرر و معیارهای مشخص شده کامپایل می کند.
- مدل را با استفاده از مجموعه داده آموزشی آموزش می دهد و آن را با استفاده از مجموعه داده اعتبارسنجی اعتبار می دهد.
- برای جلوگیری از برآش بیش از حد، توقف زودهنگام را اجرا می کند.

Sequential API Model Evaluation: مدل ترتیبی آموزش دیده را روی مجموعه داده اعتبارسنجی ارزیابی می کند و ضرر و دقت تست را چاپ می کند.

Plot Loss and Accuracy: ضرر آموزش و اعتبارسنجی و همچنین دقت را در هر دوره برای مدل ترتیبی ترسیم می کند.

ساخت و آموزش مدل Functional API:

- همان معماری CNN را با استفاده از Functional API می سازد و لایه های ورودی و خروجی را به صراحت مشخص می کند.
- شبیه به رویکرد Sequential API مدل را کامپایل، آموزش و اعتبار سنجی می کند.

- برای جلوگیری از نصب بیش از حد، توقف زودهنگام را اجرا می کند.

Functional API Model Evaluation: مدل تابعی آموزش دیده را روی مجموعه داده اعتبارسنجی ارزیابی می کند و ضرر و دقت تست را چاپ می کند.

Plot Loss and Accuracy for Functional API: مشابه مدل Sequential، این بخش از دست دادن آموزش و اعتبارسنجی و همچنین دقت را در دوره های مختلف برای مدل عملکردی ترسیم می کند.

به طور کلی، کد دو رویکرد را برای ایجاد، آموزش، ارزیابی و تجسم عملکرد مدل های CNN با استفاده از API های متوالی(Sequential) و عملکردی(Functional) ارائه شده توسط TensorFlow و Keras نشان می دهد. از یک مجموعه داده تصویری برای طبقه بندی باینری (به طور بالقوه شناسایی تومورها) استفاده می کند و نحوه کار با مجموعه داده، ساخت مدل ها، آموزش آنها و ارزیابی عملکرد آنها را نشان می دهد.

مراجع:

<https://chat.openai.com/>

<https://bard.google.com/>

<https://claude.ai/chats>

سوال پنجم

لایه های کانولوشنی در مسائلی مانند طبقه بندی تصاویر مزایای منحصر به فردی را ارائه می دهند که از روابط فضایی موجود در تصاویر استفاده می کند و منجر به بهبود عملکرد می شود. در مقابل، سناریوهای وجود دارد که این ویژگی ها ممکن است چالش هایی را ایجاد کنند مانند پردازش زبان طبیعی.

مزایای لایه های کانولوشن در طبقه بندی تصویر:

مزیت ۱: یادگیری ویژگی سلسله مراتبی فضایی(Spatial Hierarchical Feature Learning)

- مثال: طبقه بندی تصاویر

لایه های کانولوشنال می توانند ویژگی های سلسله مراتبی (لبه ها، بافت ها، الگوها) را بدون توجه به موقعیت آنها در تصویر شناسایی کنند. برای مثال، در کار طبقه بندی گریه در مقابل سگ، لایه های کانولوشن می توانند ویژگی های کلیدی مانند گوش ها، چشم های یا الگوهای پوست را بدون توجه به موقعیت آنها در تصویر شناسایی کنند. این درک فضایی به تعمیم ویژگی های آموخته شده در فضای تصویر کمک می کند و منجر به طبقه بندی قوی می شود.

همچنین ویژگی local connectivity همچنان دارد. در واقع، لایه های کانولوشنال فقط به قسمت کوچکی از تصویر ورودی متصل می شوند. این بدان معنی است که آنها می توانند یاد بگیرند که الگوهای موجود در تصویر را بدون غرق شدن در کل تصویر به یکباره شناسایی کنند.

مزیت ۲: به اشتراک گذاری پارامتر(Parameter Sharing)

- مثال: تشخیص رقم

لایه های کانولوشن وزن ها را در سراسر تصویر به اشتراک می گذارند و تعداد پارامترها را در مقایسه با شبکه های کاملاً متصل کاهش می دهند. این ویژگی به ویژه در تشخیص رقم بسیار مفید است، جایی که شبکه می تواند الگوهای مشابه (مانند منحنی ها، لبه ها) را در قسمت های مختلف تصویر تشخیص دهد. این به اشتراک گذاری پارامتر توانایی مدل را برای تعمیم و یادگیری از داده های آموزشی محدود افزایش می دهد.

مزیت ۳: Translation Invariance

- مثال: تشخیص شی

لایه های کانولوشن ویژگی Translation Invariance را دارند، به این معنی که ویژگی های آموخته شده می توانند همان الگو را بدون توجه به موقعیت آن در تصویر تشخیص دهند. در وظایف تشخیص اشیا، جایی که موقعیت یک شی ممکن است متفاوت باشد، این ویژگی شبکه را قادر می سازد تا اشیا را به رغم تغییر در مکان آنها به طور موثر تشخیص دهد.

چالش ها یا معایب لایه های هم گشتی:

چالش ۱: حساسیت به الگوهای محلی

مثال: طبقه بندی تصاویر نویزدار

- لایه های کانولوشن ممکن است بیش از حد به الگوهای محلی حساس باشند. در سناریوهایی که تصویر حاوی نویز یا ویژگی های محلی نامربوط است که می تواند شبکه را گمراه کند، لایه های کانولوشن ممکن است سهوآ روی این ویژگی ها تمرکز کنند و بر دقت طبقه بندی تأثیر منفی بگذارد.

چالش ۲: درک زمینه محدود

مثال: درک صحنه در مقیاس بزرگ

- در برخی موارد، لایه های کانولوشن ممکن است درک زمینه ای محدودی داشته باشند. به عنوان مثال، در درک صحنه در مقیاس بزرگ که درک زمینه کی (global context) بسیار مهم است، میدان پذیرنده لایه های کانولوشن ممکن است برای گرفتن اطلاعات زمینه ای گستردۀ کافی نباشد، که منجر به چالش هایی در شناخت روابط فضایی پیچیده بین اشیاء دور می شود.

چالش ۳: پیچیدگی محاسباتی

مثال: برنامه های بلاذرنگ در دستگاه های با محدودیت منابع

- لایه های کانولوشن، به ویژه در معماری های عمیق، می توانند محاسباتی فشرده داشته باشند. در برنامه های بلاذرنگ مستقر در دستگاه های محدود به منابع (به عنوان مثال، دستگاه های تلفن همراه یا سیستم های نهفته)، پیچیدگی محاسباتی لایه های کانولوشنال ممکن است چالش هایی را در دستیابی به استنتاج بلاذرنگ یا اجرای کارآمد مدل ایجاد کند.

تأثیرات بر عملکرد مدل:

- مزایای لایه های کانولوشن به بهبود دقت، استحکام و کارایی در بسیاری از کارهای طبقه بندی تصویر با اعمال نفوذ روابط فضایی، یادگیری ویژگی های سلسله مراتبی و کاهش تعداد پارامترها کمک می کند. همچنین
- با این حال، چالش های ایجاد شده توسط این لایه ها در سناریوهای خاص ممکن است منجر به کاهش دقت، تفسیر نادرست نویز، درک زمینه محدود یا افزایش پیچیدگی محاسباتی شود که بر عملکرد مدل در آن زمینه ها تأثیر منفی می گذارد. متعادل کردن مزایا و محدودیت های لایه های کانولوشن و در نظر گرفتن الزامات کاربردی خاص آنها در بهینه سازی عملکرد مدل برای وظایف طبقه بندی تصویر ضروری است.

در بالا بیشتر در کاربرد های CNN در تصویر صحبت کردیم. در ادامه دو نوع دیگر (داده های صوتی و داده های متنی) از داده ها آمده است. همان طور که در توضیحات زیر آمده است ویژگی های منحصر به فرد لایه های کانولوشنی به عملکرد قوی در داده های صوتی مانند Speech Recognition و عملکرد ضعیف در داده های متنی مانند مسائل پردازش زبان طبیعی نظری ترجمه ماشینی می شود.

چالش های لایه های کانولوشنی در NLP:

در حالی که شبکه های عصبی کانولوشنال (CNN) وظایف تشخیص تصویر را متحول کرده اند، وقتی برای برنامه های کاربردی پردازش زبان طبیعی (NLP) اعمال می شوند، با چالش های متعددی روبرو هستند. این چالش ها از تفاوت های اساسی بین تصاویر و متن ناشی می شود.

۱. فقدان نظم مکانی (Lack of Spatial Order): تصاویر دارای نظم مکانی طبیعی هستند و پیکسل ها در ساختار شبکه ای معنی دار چیده شده اند. این ساختار ذاتی به CNN ها اجازه می دهد تا به طور موثر ویژگی ها و الگوهای فضایی را استخراج کنند. با این حال،

متن فاقد این نظم ذاتی است. کلمات و عبارات به صورت متواالی مرتب شده اند و موقعیت نسبی آنها همیشه مستقیماً با معنای معنای آنها مطابقت ندارد.

۲. وابستگی های دوربرد (Long-Range Dependencies): در زبان طبیعی، معنای یک کلمه یا عبارت اغلب به بافت آن بستگی دارد که می تواند چندین کلمه با آن فاصله داشته باشد. این وابستگی های دوربرد را معرفی می کند که گرفتن آنها برای CNN دشوار است. سی ان انها معمولاً روی پنجره های متغیر محلی کار می کنند و توانایی آنها را برای درک این روابط دوربرد محدود می کنند.

۳. طول کلمات متغیر (Variable Word Lengths): برخلاف تصاویر با ابعاد پیکسل ثابت، کلمات و عبارات می توانند طول های متفاوتی داشته باشند. این تنوع، اعمال فیلترها و استخراج ویژگی های معنی دار از دنباله های متغیر با طول های مختلف را برای CNN ها چالش برانگیز می کند.

۴. عدم ترکیب بندی (Lack of Compositionality): تصاویر ذاتاً ترکیبی هستند، به این معنی که معنای تصویر از چینش و روابط اجزای تشکیل دهنده آن به دست می آید. در مقابل، متن همیشه ترکیبی نیست. معنای یک جمله ممکن است صرفاً مجموع معانی تک تک کلمات آن نباشد، بلکه بیشتر به تعاملات پیچیده بین کلمات بستگی دارد.

۵. پراکندگی داده ها (Data Sparsity): وظایف NLP اغلب با واژگان وسیعی از کلمات سروکار دارد که منجر به نمایش داده های پراکنده می شود. این پراکندگی می تواند مانع آموخته CNN ها شود، زیرا آنها به حجم زیادی از داده های متراکم برای یادگیری موثر الگوهای معنادار نیاز دارند.

با توجه به این چالش ها، CNN ها به اندازه ای که در تشخیص تصویر بوده اند، در NLP مورد استفاده قرار نگرفته اند. معماری های جایگزین، مانند شبکه های عصبی بازگشتی (RNN) و ترانسفورماتورها، برای کارهای NLP مؤثرتر بوده اند، زیرا برای مدیریت ماهیت متواالی زبان و وابستگی های دوربرد مناسب تر هستند.

مزیت های استفاده از لایه های کانولوشنی در پردازش صوت و speech recognition

شبکه های عصبی کانولوشنال (CNN) می توانند در پردازش صدا و وظایف تشخیص گفتار استفاده شوند. در واقع، CNN ها در سال های اخیر به دلیل توانایی آنها در استخراج الگوهای پیچیده از داده های صوتی، به طور فزاینده ای محبوب شده اند.

CNN ها به ویژه برای وظایف تشخیص گفتار مناسب هستند زیرا می توانند به طور موثر ویژگی های زمانی و طیفی سیگنال های گفتار را ضبط کنند. ویژگی های زمانی مربوط به ترتیبی است که صدایها در طول زمان رخ می دهند، در حالی که ویژگی های طیفی به محتوای فرکانس صدایها مربوط می شود.

با تجزیه و تحلیل ویژگی های زمانی و طیفی، CNN ها می توانند الگوهای منحصر به فردی را که با واج ها، کلمات و عبارات مختلف مطابقت دارند، شناسایی کنند. این آنها را به ابزاری مؤثر برای شناسایی گوینده و رونویسی گفتار به متن تبدیل می کند.

در اینجا برخی از مزایای استفاده از CNN در پردازش صدا و تشخیص گفتار آورده شده است:

- دقت بالا: CNN ها می توانند در کارهای تشخیص گفتار، حق در محیط های پرس و صدا، به دقت بالایی دست یابند.
- استحکام در برابر نویز: CNN ها نسبتاً در برابر نویز مقاوم هستند، که برای برنامه های تشخیص گفتار در دنیای واقعی مهم است.
- توانایی یادگیری الگوهای پیچیده: CNN ها می توانند الگوهای پیچیده را از داده های صوتی بیاموزند، که آنها را برای کارهای مانند شناسایی بلندگو و تشخیص احساسات مناسب می کند.

با این حال، برخی از چالش های مرتبط با استفاده از CNN در پردازش صدا و تشخیص گفتار نیز وجود دارد:

- پیچیدگی محاسباتی: CNN ها می توانند از نظر محاسباتی برای آموخته و اجراگران باشند.
- الزامات داده: CNN ها برای دستیابی به دقت بالا به حجم زیادی از داده های آموخته نیاز دارند.
- مشکل در مدیریت وابستگی های دوربرد: CNN ها می توانند در مدیریت وابستگی های دوربرد در سیگنال های گفتاری مشکل داشته باشند.

با وجود این چالش‌ها، CNN‌ها به ابزار مهندی برای پردازش صدا و تشخیص گفتار تبدیل شده‌اند. با ادامه تحقیقات، این احتمال وجود دارد که CNN‌ها در این وظایف حتی موثرتر شوند.

در اینجا چند نمونه از نحوه استفاده از CNN در پردازش صدا و تشخیص گفتار (voice processing and speech recognition) آورده شده است:

- تشخیص خودکار گفتار (ASR): CNN‌ها برای توسعه سیستم‌های ASR استفاده می‌شوند که می‌توانند گفتار را به متن رونویسی کنند.
- شناسایی سخنگو: CNN‌ها برای توسعه سیستم‌های استفاده می‌شوند که می‌توانند سخنگوها را بر اساس صدایشان شناسایی کنند.
- تشخیص احساسات: CNN‌ها برای توسعه سیستم‌های استفاده می‌شوند که می‌توانند احساسات را از گفتار تشخیص دهند.
- تشخیص فعالیت صوتی (VAD): CNN‌ها برای توسعه سیستم‌های استفاده می‌شوند که می‌توانند حضور گفتار را در ضبط‌های صوتی تشخیص دهند.

به طور کلی، CNN‌ها ابزار قدرتمندی برای پردازش صدا و تشخیص گفتار هستند. آنها این پتانسیل را دارند که دقیق و استحکام این وظایف را بهبود بخشدند و آنها را در برنامه‌های کاربردی دنیای واقعی مفیدتر کنند.

مراجع:

<https://chat.openai.com/>

<https://bard.google.com/>

<https://claude.ai/chats>

سوال ششم

فرض کنید یک لایه کانولوشنی داریم که یک تنسور شکل به (N, F, H, W) را خروجی می‌دهد که در آن: N اندازه دسته است.

F تعداد فیلترهای کانولوشن است.
 H ابعاد فضایی هستند.

فرض کنید این خروجی به عنوان ورودی به یک لایه کانولوشنی با 1×1 فیلترهای 1×1 ، padding ۰ و گام ۱ وارد شده است. سپس خروجی این لایه کانولوشنی 1×1 شکل $(N, F, 1, H, W)$ خواهد داشت. بنابراین می‌توان از تعداد فیلترهای 1×1 لایه کانولوشنی برای تغییر ابعاد در فضای فیلتر استفاده کرد. اگر $F > 1$ در حال افزایش ابعاد هستیم، اگر $F < 1$ ابعاد را کاهش می‌دهیم.

الف) هدف از فیلترهای 1×1 :

فیلترهای 1×1 که به عنوان کانولوشن نقطه‌ای (pointwise convolution, often called feature map pooling or a projection layer) نیز شناخته می‌شوند، نوعی فیلتر کانولوشن هستند که اندازه هسته آن 1×1 است. این بدان معناست که فیلتر یک وزن واحد را به هر پیکسل در نقشه ویژگی ورودی اعمال می‌کند و در نتیجه یک مقدار خروجی برای هر پیکسل ایجاد می‌کند. همچنین به آنها channel-wise pooling گفته می‌شود.

هدف اصلی استفاده از فیلترهای 1×1 کاهش تعداد نقشه‌های ویژگی با حفظ ویژگی‌های مهم است. این به این دلیل است که فیلترهای 1×1 هیچ اطلاعات مکانی جدیدی معرفی نمی‌کنند، اما می‌توان از آنها برای اصلاح نقشه‌های ویژگی موجود به روش‌های مختلف استفاده کرد.

به عنوان مثال، فیلترهای 1×1 را می‌توان برای موارد زیر استفاده کرد:

- کاهش ابعاد نقشه های ویژگی: آنها تعداد نقشه های ویژگی (عمق) را کاهش می دهند و در عین حال ویژگی های اساسی را حفظ می کنند. با استفاده از کanal های خروجی کمتر از ورودی، آنها به مدیریت پیچیدگی محاسباتی و کنترل پارامترهای مدل کمک می کنند.
- انجام تبدیل های خطی بر روی نقشه های ویژگی: این می تواند برای بهبود نمایش ویژگی برای لایه های بعدی استفاده شود. معرفی غیر خطی بودن: این را می توان با اعمال یک تابع فعال سازی، مانند ReLU ، در خروجی کانولوشن 1×1 انجام داد. این کار به در بر گرفتن الگوهای پیچیده یا روابط بین کanal ها کمک می کند.
- Projecting Feature Maps: تعداد نقشه های ویژگی ایجاد شده ثابت خواهد بود و اثر آن ممکن است اصلاح ویژگی های باشد که قبلاً استخراج شده اند. این معمولاً برخلاف ادغام ویژگی های سنتی در هر کanal، ادغام کanalی (channel-wise pooling) نامیده می شود. هیچ تغییری در عرض یا ارتفاع نقشه های ویژگی ایجاد نمی شود و با طراحی، تعداد نقشه های ویژگی با اعمال یک عملیات طرح ریزی ساده ثابت نگه داشته می شود.
- افزایش ابعاد نقشه های ویژگی: همچنین اگر تعداد فیلتر های لایه کانولوشن 1×1 از عمق ورودی بیشتر باشد منجر به افزایش ابعاد نقشه های ویژگی می شود. این یک عملیات متداول است که بعد از یک لایه ادغام قبل از اعمال لایه کانولوشن دیگری استفاده می شود. اثر projection فیلتر می تواند به تعداد دفعات مورد نیاز روى ورودی اعمال شود، و به شما امکان می دهد تعداد نقشه های ویژگی را بزرگتر کنید و در عین حال ترکیبی داشته باشید که ویژگی های برجسته اصلی را به تصویر بکشد.

ب) اطلاعات ارائه شده توسط Feature Map 1×1
 نقشه ویژگی 1×1 خلاصه ای از اطلاعات را از نقشه ویژگی ورودی مربوطه ارائه می دهد. این نقشه ویژگی شامل مقادیری است که هر یک از آنها مجموع وزن دار مقادیر موجود در کanal های یک پیکسل در نقشه ویژگی ورودی را نشان می دهد. همچنین ممکن است پس از مجموع وزن دار تابع فعال ساز غیر خطی اعمال شود که پیچیدگی را بیشتر کند و الگوهای پیچیده تری را به ارمغان آورد.

این نمایش خلاصه می تواند برای ثبت ویژگی های مهم و کاهش نویز مفید باشد. به عنوان مثال، اگر یک نقشه ویژگی ورودی حاوی اطلاعاتی در مورد لبه های یک شی باشد، نقشه ویژگی 1×1 ممکن است فقط حاوی اطلاعاتی در مورد وجود یا عدم وجود لبه باشد.

به طور خلاصه:

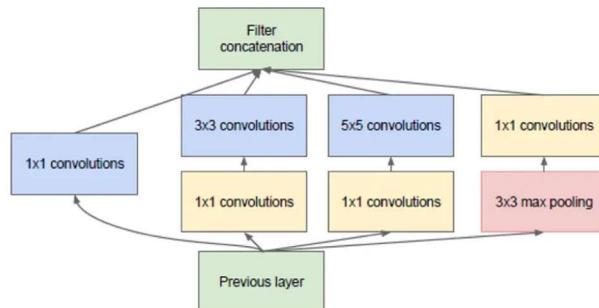
- کانولوشن های 1×1 تبدیل خطی نقشه های ویژگی ورودی را با استفاده از اندازه هسته 1×1 انجام می دهند. می توان با استفاده از تابع فعال سازی مانند ReLU آن را غیر خطی نیز کنیم.
- هر پیکسل خروجی مجموع وزنی پیکسل های ورودی مربوطه است.
- فیلتر 1×1 به عنوان ترکیبی خطی از کanal های ورودی عمل می کند و نمایش های جدیدی از نظر کanal ایجاد می کند.

(پ)
 نقشه ویژگی 1×1 با تصویر اصلی یا فیلترهای دیگر با اندازه های مختلف متفاوت است، زیرا یک linear projection (و همچنین با اعمال تابع فعال سازی non-linear projection) از مجموعه ای از نقشه های ویژگی است. پروژکشن ایجاد شده توسط یک 1×1 می تواند مانند ادغام کanal (channel-wise pooling) عمل کند و برای کاهش ابعاد استفاده شود. projection ایجاد شده توسط یک 1×1 همچنین می تواند به طور مستقیم استفاده شود یا برای افزایش تعداد نقشه های ویژگی در یک مدل استفاده شود.

تفاوت با تصویر اصلی: نقشه های ویژگی 1×1 با تصویر اصلی متفاوت هستند زیرا نسخه های تغییر یافته ورودی را از طریق مجموع وزن دار یا همان ترکیب خطی (یا غیر خطی با اعمال تابع فعال سازی) کanal ها نشان می دهند. آنها به جای روابط فضایی (spatial relationships) بر روابط بین کanal (channel-wise relationships) تمرکز می کنند. در حالی که مقادیر در تصویر اصلی نشان دهنده شدت پیکسل خام است.

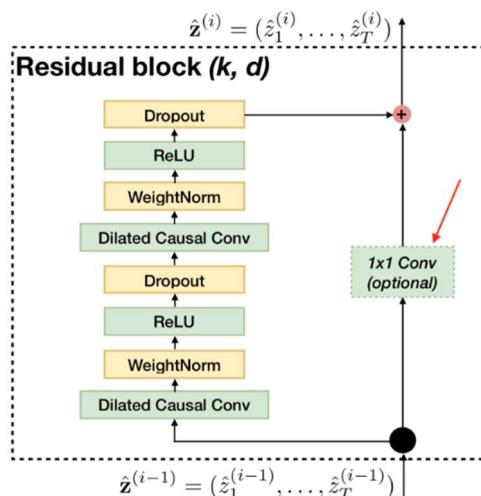
تفاوت با فیلترهای دیگر: در مقایسه با فیلترهای با اندازه بزرگتر، فیلترهای 1×1 اطلاعات مکانی را در بر نمی گیرند، بلکه بر تغییر ابعاد (کاهش و افزایش) در کanal ها تمرکز می کنند. این بدان معنی است که فیلترهای 1×1 نسبت به تغییر موقعیت اشیا در تصویر ورودی حساسیت کمتری دارند.

ت) مدل های با استفاده از فیلترهای 1×1 : شبکه Inception یکی از اولین مدل هایی بود که از فیلترهای 1×1 به طور گسترده استفاده کرد. از فیلترهای 1×1 برای کاهش ابعاد نقشه های ویژگی و انجام تبدیل های خطی استفاده می کند.

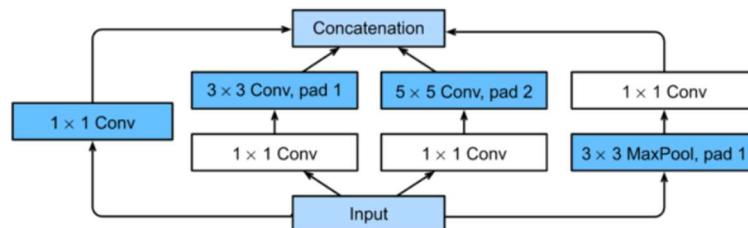


(b) Inception module with dimensionality reduction

ResNet: بعضی از معماری های ResNet همچنین از فیلترهای 1×1 به طور گسترده استفاده می کند. از فیلترهای 1×1 برای انجام نگاشت هويت (identity mappings) استفاده می کند که به شبکه اجازه می دهد تا اتصالات باقی مانده عمیق تری (deeper residual connections) را بیاموزد. همچنین برای کاهش ابعاد فضایی با استفاده از stride هم از آن استفاده می شود تا بتوانیم جمع را انجام دهیم. معماری های ResNet از فیلترهای 1×1 در بلوک های باقی مانده (residual blocks) برای تبدیل های کانال استفاده می کنند، که به یادگیری نمایش های پیچیده به طور موثر کمک می کند.

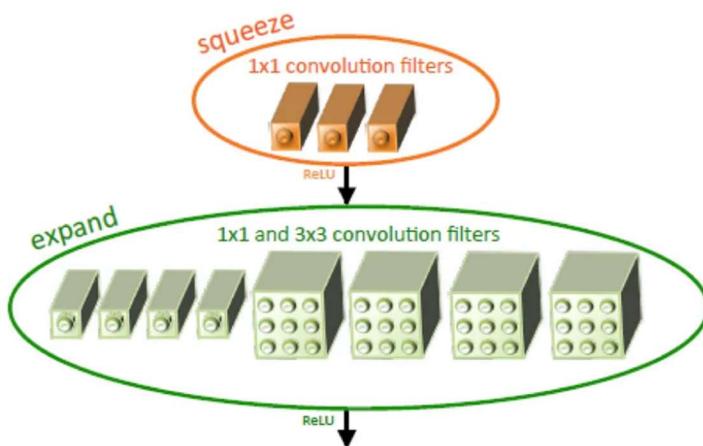


GoogleNet: GoogleNet از فیلترهای 1×1 (همان مازول inception) برای انجام کاهش ابعاد، کاهش تعداد پارامتر های شبکه و ترکیب موثر اطلاعات از مقیاس های مختلف استفاده می کند.



ساختار شبکه SqueezeNet از Fire Modules و Expansion Layer های خودکار تشکیل شده است به طور گسترده ای از فیلتر های 1×1 استفاده می کند. "Fire Modules" که شامل:

۱. Squeeze Layer که فقط فیلترهای 1×1 دارد.
 ۲. این یک لایه Expansion را تغذیه می کند که دارای ترکیبی از فیلترهای 1×1 و 3×3 است.
 ۳. تعداد فیلترها در Squeeze Layer کمتر از تعداد فیلترهای 1×1 + تعداد 3×3 در Expand Layer تنظیم شده است.
- در حال حاضر مشخص است که فیلترهای 1×1 در Squeeze Layer چه می کنند - آنها تعداد پارامترها را با "down-sampling" کانال های ورودی قبل از وارد شدن به لایه Expand کاهش می دهند.



ث) شرایطی که فیلترهای 1×1 ممکن است مفید نباشند:

- تصویر یا کارهایی که نیاز به درک مکانی دقیق دارند، استفاده از فیلترهای 1×1 ممکن است به اندازه فیلترهای بزرگ تر که زمینه فضایی (spatial context) را ثابت می کنند مؤثر نباشد.
- Limited Channel Interactions: اگر شبکه به تعامل زیاد بین کانال ها نیاز دارد یا اگر داده ها از کاهش ابعاد کانال سود نمی برد، استفاده از فیلترهای 1×1 ممکن است مزایای قابل توجهی را ارائه نکند.
- علاوه بر این، اگر داده های ورودی در حال حاضر بسیار پراکنده (sparse) هستند، استفاده از فیلترهای 1×1 ممکن است مفید نباشد. این به این دلیل است که فیلترهای 1×1 می توانند پراکنندگی داده ها را بیشتر کنند که می تواند یادگیری شبکه را دشوار کند.
- اگر داده های ورودی از قبل ابعاد پایینی دارند، استفاده از فیلترهای 1×1 ممکن است هیچ مزیتی نداشته باشد. علاوه بر این، اگر مدل در حال حاضر کوچک است، استفاده از فیلترهای 1×1 ممکن است کاهش قابل توجهی در تعداد پارامترها ایجاد نکند.

ج) پیاده سازی یک مدل کانولوشنی ساده با فیلتر 1×1 :

مدلی برای افزایش ابعاد کانال با استفاده از فیلتر کانولوشنی 1×1 :

```
import tensorflow as tf

model = tf.keras.Sequential([
    tf.keras.layers.Input(shape=(28, 28, 3)),
    # Add a layer with 16 channels and 3x3 kernel
    tf.keras.layers.Conv2D(16, (3, 3), activation='relu'),
    # Add a layer with 32 channels and 3x3 kernel
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu'),
    # Add a layer with 64 channels and 3x3 kernel
    tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
    # Add a layer with 128 channels and 3x3 kernel
    tf.keras.layers.Conv2D(128, (3, 3), activation='relu'),
    # Add a layer with 256 channels and 3x3 kernel
    tf.keras.layers.Conv2D(256, (3, 3), activation='relu'),
    # Add a layer with 512 channels and 3x3 kernel
    tf.keras.layers.Conv2D(512, (3, 3), activation='relu'),
    # Add a layer with 1024 channels and 3x3 kernel
    tf.keras.layers.Conv2D(1024, (3, 3), activation='relu'),
    # Add a layer with 2048 channels and 3x3 kernel
    tf.keras.layers.Conv2D(2048, (3, 3), activation='relu'),
    # Add a layer with 4096 channels and 3x3 kernel
    tf.keras.layers.Conv2D(4096, (3, 3), activation='relu'),
    # Add a layer with 4096 channels and 3x3 kernel
    tf.keras.layers.Conv2D(4096, (3, 3), activation='relu'),
    # Add a layer with 1024 channels and 3x3 kernel
    tf.keras.layers.Conv2D(1024, (3, 3), activation='relu'),
    # Add a layer with 512 channels and 3x3 kernel
    tf.keras.layers.Conv2D(512, (3, 3), activation='relu'),
    # Add a layer with 256 channels and 3x3 kernel
    tf.keras.layers.Conv2D(256, (3, 3), activation='relu'),
    # Add a layer with 128 channels and 3x3 kernel
    tf.keras.layers.Conv2D(128, (3, 3), activation='relu'),
    # Add a layer with 64 channels and 3x3 kernel
    tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
    # Add a layer with 32 channels and 3x3 kernel
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu'),
    # Add a layer with 16 channels and 3x3 kernel
    tf.keras.layers.Conv2D(16, (3, 3), activation='relu'),
    # Add a layer with 8 channels and 3x3 kernel
    tf.keras.layers.Conv2D(8, (3, 3), activation='relu'),
    # Add a layer with 4 channels and 3x3 kernel
    tf.keras.layers.Conv2D(4, (3, 3), activation='relu'),
    # Add a layer with 2 channels and 3x3 kernel
    tf.keras.layers.Conv2D(2, (3, 3), activation='relu'),
    # Add a layer with 1 channel and 3x3 kernel
    tf.keras.layers.Conv2D(1, (3, 3), activation='relu')
])
```

```

# 3x3 convolution which increases the number of channels from 3 to 16
tf.keras.layers.Conv2D(filters=16, kernel_size=(1, 1),
activation='relu'), # activation -> non-linear (relu)
# tf.keras.layers.Conv2D(filters=16, kernel_size=(1, 1)), # No activation
-> linear

tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
tf.keras.layers.Flatten(),
tf.keras.layers.Dense(10, activation='softmax')
])

model.summary()

```

Model: "sequential"		
Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 28, 28, 16)	64
max_pooling2d (MaxPooling2 D)	(None, 14, 14, 16)	0
flatten (Flatten)	(None, 3136)	0
dense (Dense)	(None, 10)	31370

Total params: 31434 (122.79 KB)
Trainable params: 31434 (122.79 KB)
Non-trainable params: 0 (0.00 Byte)

این مدل دارای یک لایه کانولوشن با فیلتر 1×1 می باشد. شکل ورودی $(28, 28, 3)$ و شکل خروجی $(28, 28, 16)$ است. پس از اعمال maxPooling با اندازه $(2, 2)$ ، شکل خروجی $(14, 14, 16)$ می شود. در نهایت، خروجی صاف(flat) می شود و از یک لایه متراکم با 10 واحد و تابع فعال سازی softmax عبور می کند.

مدلی برای کاهش ابعاد کanal با استفاده از فیلتر کانولوشنی 1×1 :

```

import tensorflow as tf

model = tf.keras.Sequential([
    tf.keras.layers.Input(shape=(28, 28, 3)),

    # 1x1 convolution which reduces the number of channels from 3 to 1
    tf.keras.layers.Conv2D(filters=1, kernel_size=(1, 1), activation='relu'),
    # activation -> non-linear (relu)
    # tf.keras.layers.Conv2D(filters=1, kernel_size=(1, 1)), # No activation
    -> linear

    tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),

```

```

    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(10, activation='softmax')
])

model.summary()

```

Model: "sequential_1"		
Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 28, 28, 1)	4
max_pooling2d_1 (MaxPooling2D)	(None, 14, 14, 1)	0
flatten_1 (Flatten)	(None, 196)	0
dense_1 (Dense)	(None, 10)	1970
<hr/>		
Total params: 1974 (7.71 KB)		
Trainable params: 1974 (7.71 KB)		
Non-trainable params: 0 (0.00 Byte)		

این مدل دارای یک لایه کانولوشن با فیلتر 1×1 می باشد. شکل ورودی $(28, 28, 3)$ و شکل خروجی $(1, 28, 28)$ است. پس از اعمال maxPooling با اندازه $(2, 2)$ ، شکل خروجی $(14, 14, 1)$ می شود. در نهایت، خروجی صاف(flat) می شود و از یک لایه متراکم با 10 واحد و تابع فعال سازی softmax عبور می کند.

همان طور که در بالا مشاهده می شود وقتی که تعداد فیلتر های لایه کانولوشنی 1×1 کمتر از کانال های ورودی است ($\text{filters}=1$) با کاهش تعداد کانال ها و پارامتر ها موجه می شویم و در نهایت 1974 پارامتر خواهیم داشت. ولی وقتی که تعداد فیلتر های لایه کانولوشنی 1×1 بیشتر از کانال های ورودی است ($\text{filters}=32$) با افزایش تعداد کانال ها و پارامتر ها موجه می شویم و در نهایت 31434 پارامتر خواهیم داشت. همچنین ما از تابع فعال ساز `relu` نیز استفاده کردیم تا non-linearity را به شبکه معرف کنیم ولی در زیر آن کدی که کامنت شده است از آن استفاده نمی کند و در بعضی سناریو ها فقط نیاز به ترکیب خطی داریم.

تأثیر یک لایه کانولوشنی 1×1 روی یک ورودی دلخواه برای کاهش ابعاد:

```

✓ 0s ➔ import tensorflow as tf

model = tf.keras.Sequential([
    tf.keras.layers.Input(shape=(28, 28, 3)),
    tf.keras.layers.Conv2D(filters=1, kernel_size=(1, 1), activation='relu'),
])

print("image shape before applying model:", my_image.shape)
new_image = model(my_image)
print("-----")
print("image shape before applying model:", new_image.shape)

➡️ image shape before applying model: (1, 28, 28, 3)
-----
image shape before applying model: (1, 28, 28, 1)

```

تاثیر یک لایه کانولوشنی 1×1 روی یک ورودی دلخواه برای افزایش ابعاد:

```

✓ 0s ➔ import tensorflow as tf

model = tf.keras.Sequential([
    tf.keras.layers.Input(shape=(28, 28, 3)),
    tf.keras.layers.Conv2D(filters=16, kernel_size=(1, 1), activation='relu'),
])

print("image shape before applying model:", my_image.shape)
new_image = model(my_image)
print("-----")
print("image shape before applying model:", new_image.shape)

➡️ image shape before applying model: (1, 28, 28, 3)
-----
image shape before applying model: (1, 28, 28, 16)

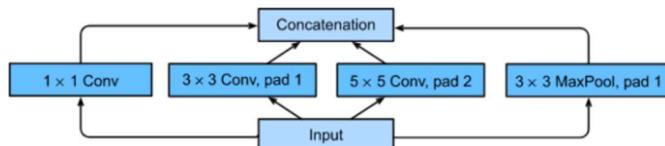
```

:مراجع

<https://chat.openai.com/>
<https://bard.google.com/>
<https://claude.ai/chats>
<https://machinelearningmastery.com/introduction-to-1x1-convolutions-to-reduce-the-complexity-of-convolutional-neural-networks/>
[1X1 Convolution, CNN, CV, Neural Networks | Analytics Vidhya \(medium.com\)](1X1 Convolution, CNN, CV, Neural Networks | Analytics Vidhya (medium.com))

GoogLeNet

- شبکه GoogLeNet برنده مسابقه ILSVRC'14 با خطای ۶.۷٪ شد
- یکی از تمرکزهای مقاله پرداختن به این سوال بود که بهترین اندازه برای کرنل‌های کانولوشنی چند است؟
- فیلترهای هم‌عرض (موازی) تحت عنوان Inception Module معرفی شدند
 - کانولوشن‌های دارای ابعاد مختلف و ادغام
 - سپس، خروجی تمام فیلترها به هم می‌شوند (در عمق)
- در این ساختار عمق نقشه‌ها حتماً زیاد می‌شود

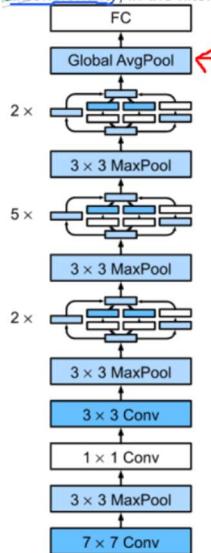


۱۹

Szegedy, Christian, et al. "Going deeper with convolutions." CVPR, 2015.

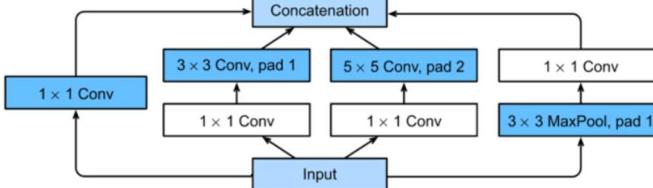
Suppose that I have a conv layer which outputs an (N, F, H, W) shaped tensor where:
 N is the batch size. F is the number of convolutional filters. H,W are the spatial dimensions
 Suppose the input is fed into a conv layer with $F_1 \times 1$ filters, zero-padding and stride 1. Then the output of this 1×1 conv layer will have shape (N, F_1, H, W) . So 1×1 conv filters can be used to change the dimensionality in the filter space. If $F_1 > F$ then we are increasing dimensionality, if $F_1 < F$ we are decreasing dimensionality, in the filter dimension.

GoogLeNet



- برای کاهش عمق و ترکیب آنها می‌توان از فیلترهای 1×1 استفاده کرد
- در معماری GoogLeNet از چندین بلوک Inception استفاده شده است
- برای بهبود جریان گرادیان، دو دسته‌بند کمکی در لایه‌های میانی شبکه قرار داده شده بود که امروزه با توجه به الگوریتم‌های یادگیری جدیدتر چندان ضروری نیست
- به منظور کاهش پارامترها، از GAP استفاده کرده است

- در این مثال ادغام 7×7



ساختار و هدف مازول Inception را در این شبکه توضیح دهید.

مازول Inception که در معماری GoogLeNet معرفی شده است، جزء کلیدی شبکه‌های عصبی کانولوشنال مدرن (CNN) است. این طراحی شده است تا ویژگی‌ها را در مقیاس‌های فضایی چندگانه به طور کارآمد ثبت کند، و در عین حال که هزینه‌های محاسباتی را به حداقل می‌رساند، قدرت نمایش شبکه را افزایش می‌دهد. مازول Inception از عملیات کانولوشنی موازی با اندازه‌های مختلف هسته و ادغام استفاده می‌کند و شبکه را قادر می‌سازد تا ویژگی‌های متنوعی را بیاموزد.

ساختار مژول inception:

ماژول Inception شامل چندین شاخه است که هر کدام عملیات متفاوت را روی نقشه های ویژگی ورودی انجام می دهند. این شاخه ها به صورت موازی اجرا می شوند و قبل از ارسال خروجی به لایه بعدی به هم متصل می شوند. ساختار معمولاً شامل:

۱. شاخه کانولوشن ۱*:۱

این شاخه از یک لایه کانولوشن ۱*۱ تشکیل شده است که کاهش ابعاد را انجام می دهد. این روابط کanal عاقلانه را ضبط می کند و به کاهش تعداد کanal های ورودی کمک می کند و به عنوان یک لایه گلوگاه عمل می کند.

۲. شاخه کانولوشن ۳*:۳

این شاخه شامل یک کانولوشن ۱*۱ به دنبال یک لایه کانولوشن ۳*۳ است. این اطلاعات فضایی را در یک میدان پذیرای محلی ضبط می کند و ویژگی ها و الگوهای سطح متوسط را یاد می گیرد.

۳. شاخه کانولوشن ۵*:۵

مشابه شاخه ۳*۳، این شاخه از یک کانولوشن ۱*۱ و به دنبال آن یک لایه کانولوشنال ۵*۵ استفاده می کند. این ویژگی های فضایی بزرگ تر را به تصویر می کشد و چشم انداز وسیع تری از ورودی را به شبکه ارائه می دهد.

۴. شاخه ادغام:

این شاخه حداقل عملیات ادغام را برای ثبت الگوهای فضایی در مقیاس متفاوت انجام می دهد. ابعاد فضایی ورودی را کاهش می دهد و به عنوان مکمل شاخه های کانولوشنال عمل می کند.

هدف و مزایای مژول inception:

استخراج ویژگی چند مقیاسی:

هدف اصلی مژول Inception گرفتن ویژگی ها در مقیاس های فضایی چندگانه به طور همزمان است. با استفاده از اندازه های مختلف هسته (1*1, 3*3, 5*5) و عملیات ادغام موازی، شبکه را قادر می سازد تا ویژگی ها را در سطوح مختلف انتزاع بیاموزد و اطلاعات دقیق و کلی را به دست آورد.

پردازش اطلاعات کارآمد:

ماژول Inception اطلاعات را از زمینه های مختلف دریافتی به طور موثر ترکیب می کند. این به شبکه اجازه می دهد تا مجموعه متنوعی از ویژگی ها را بدون افزایش پیچیدگی محاسباتی قابل توجهی بیاموزد، زیرا عملیات موازی پارامترها را به اشتراک می گذارد و می تواند همزمان محاسبه شود.

افزایش ظرفیت شبکه:

با ترکیب انواع مختلف کانولوشن و عملیات ادغام در یک مژول واحد، معماری Inception ظرفیت نمایش شبکه را افزایش می دهد. این به گرفتن الگوهای پیچیده کمک می کند و توانایی شبکه را برای یادگیری ویژگی های متمایز بهبود می بخشد.

استفاده مجدد از ویژگی و عمق شبکه:

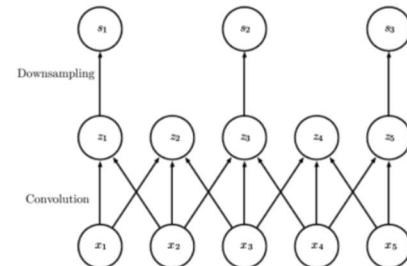
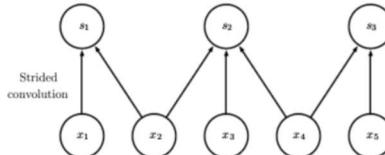
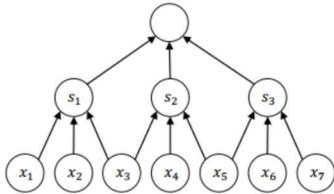
استفاده از چندین عملیات به صورت موازی امکان استفاده مجدد از ویژگی ها را در همان لایه فراهم می کند و معماری های عمیق تر را بدون افزایش بیش از حد تعداد پارامترها تسهیل می کند.

به طور خلاصه، مژول Inception به عنوان یک بلوک سازنده کارا در معماری های CNN عمل می کند، استخراج ویژگی های چند مقیاسی را تسهیل می کند، کارایی پردازش اطلاعات را بهبود می بخشد و ظرفیت شبکه را برای یادگیری بازنمایی های متنوع و آموختن از داده های ورودی افزایش می دهد. طراحی ساختاریافته آن باعث ایجاد تعادل بین کارایی محاسباتی و عملکرد شبکه می شود و به موفقیت معماری های مدرن CNN در وظایف مختلف بینایی کامپیوتر کمک می کند.

چگونه اندازه پارامتر گام در لایه های هم گشته بر ابعاد فضایی نگاشت ویژگی ها تأثیر می گذارد؟

گام (Stride)

- برای کاهش هزینه محاسباتی می توانیم از برخی موقعیت ها پرش کنیم
 - به قیمت استخراج نشدن ویژگی ها با رزولوشن کامل
 - به نوعی **downsampling** انجام می شود
 - باعث افزایش میدان تأثیر می شود



۱۸

پارامتر گام در لایه های کانولوشن، نحوه حرکت فیلتر/هسته به صورت فضایی در داده های ورودی (مانند یک تصویر) را کنترل می کند و مستقیماً بر ابعاد فضایی نقشه های ویژگی تولید شده توسط عملیات کانولوشن تأثیر می گذارد. اندازه گام تعیین کننده میزان تغییر یا حرکت اعمال شده به فیلتر/هسته در حین کانولوشن آن بر روی ورودی است.

تأثیر اندازه گام بر ابعاد فضایی:

- گام = ۱ (پیش فرض):
 - هنگامی که گام روی ۱ تنظیم می شود، فیلتر/هسته هر بار یک پیکسل در هر دو جهت افقی و عمودی در سراسر ورودی حرکت می کند.
 - نقشه های ویژگی خروجی دارای ابعاد مکانی (عرض و ارتفاع) یکسانی با ورودی هستند، زیرا هر کاربرد فیلتر یک عنصر خروجی تولید می کند و در نتیجه اندازه فضایی کاهش نمی یابد. (در صورت استفاده از padding اگر از padding استفاده نکنیم ابعاد مکانی به انداز سایز کرنل منهای یک کاهش می یابند).
- گام > ۱:
 - هنگامی که گام بزرگتر از ۱ باشد (به عنوان مثال، گام = ۲)، فیلتر/هسته هر بار بیش از یک پیکسل در سراسر ورودی حرکت می کند.
 - یک گام بزرگتر باعث می شود که فیلتر در حین پیچیدگی پیکسل ها را رد کند و در نتیجه باعث کاهش همپوشانی بین فیلدهای گیرنده و عناصر خروجی کمتر شود.
 - ابعاد فضایی نقشه ویژگی خروجی کاهش می یابد که منجر به کاهش نمونه گیری فضایی یا کاهش ابعاد فضایی می شود. به نوعی **downsampling** انجام می شود. (ویژگی ها با رزولوشن کامل استخراج نم شوند).
 - همچنین باعث افزایش میدان تأثیر (receptive field) در لایه ها می شود.

Formulation for Spatial Dimension Calculation:

Given:

- Input spatial dimensions (W_{in} , H_{in})
- Filter size (F)
- Stride (S)
- Padding (P)

The formula to calculate the output spatial dimensions (W_{out} , H_{out}) after applying convolution with stride and padding is given by:

$$W_{out} = \frac{W_{in}-F+2P}{S} + 1$$
$$H_{out} = \frac{H_{in}-F+2P}{S} + 1$$

Where:

- W_{in} and H_{in} are the input width and height.
- W_{out} and H_{out} are the output width and height.
- F is the filter size/kernel size.
- S is the stride length.
- P is the amount of padding applied to the input.

خلاصه:

- یک مقدار گام بزرگتر باعث می شود که فیلتر/هسته سریعتر در ورودی حرکت کند، که منجر به کاهش ابعاد فضایی خروجی می شود. همچنین منجر به افزایش میدان تاثیر (receptive field) می شود.
- Stride بوضوح فضای نقشه های ویژگی در CNN تاثیر می گذارد. گام های بالاتر منجر به نمونه برداری تهاجمی تر، کاهش اطلاعات مکانی و نمایش سلسه مراتبی بالقوه در لایه های بعدی می شود.
- انتخاب یک مقدار گام مناسب بسیار مهم است و به مبالغه مورد نظر بین وضوح فضایی و کارایی محاسباتی در یک معماری یا کار شبکه عصبی معین بستگی دارد.

ویژگی های کلیدی و عملیات لایه های هم گشته مورد استفاده در شبکه خود و اهمیت آنها در استخراج ویژگی را شرح دهید.

در این قسمت با توجه به ساختار مدل ۲ به توضیح می پردازیم.

شرح لایه های کانولوشنال:

Conv1 با ۳۲ فیلتر اندازه (۳، ۳)

عملکرد: این لایه کانولوشن ۳۲ فیلتر، هر کدام با اندازه ۳*۳ را به ورودی اعمال می کند.

گام: گامها روی (۱، ۱) تنظیم می شوند که نشان می دهد فیلتر هر بار یک پیکسل به صورت افقی و عمودی در ورودی حرکت می کند. اعمال می شود و اطمینان حاصل می شود که نقشه های ویژگی خروجی همان ابعاد فضایی ورودی را دارند.

فعال سازی: تابع فعال سازی واحد خطی اصلاح شده (ReLU) پس از عملیات کانولوشن استفاده می شود.

اهمیت در استخراج ویژگی: این لایه ۳۲ نقشه ویژگی های مختلف را استخراج می کند که هر کدام الگوها، لبه ها یا بافت های متفاوت را از تصویر ورودی می گیرد. با استفاده از اندازه های کوچک فیلتر (۳*۳) و فعال سازی ReLU، به یادگیری موثر ویژگی های محلی کمک می کند.

Conv2D با ۶۴ فیلتر اندازه (۳، ۳) و Stride (۲، ۲)

عملیات: این لایه کانولوشن ۶۴ فیلتر به اندازه ۳*۳ را روی نقشه های ویژگی ورودی به دست آمده از لایه قبلی (conv1) اعمال می کند.

گام: گام‌ها روی (۲، ۲) تنظیم می‌شوند که باعث می‌شود فیلتر در یک زمان دو پیکسل را به صورت افقی و عمودی حرکت دهد.
Padding: از "padding=same" دوباره برای حفظ ابعاد فضایی استفاده می‌شود.

فعال سازی: تابع فعال سازی ReLU پس از عملیات کانولوشن اعمال می‌شود.
اهمیت در استخراج ویژگی: Conv2 بیشتر ۶۴ نقشه ویژگی متمایز را از خروجی Conv1 استخراج می‌کند، اما با کاهش ابعاد فضایی به دلیل گام برداشتن (۲، ۲). گام بزرگتر منجر به down-sampling می‌شود و ویژگی‌های سطح بالاتری را که زمینه‌های پذیرنده بزرگتر را پوشش می‌دهند، به تصویر می‌کشد.

:Inception Module

یک تابع پیاده شده است که شامل چندین عملیات کانولوشنال موازی (مشابه ساختار ماقول Inception که قبلاً بحث شد) برای ثبت ویژگی‌ها در مقیاس‌های چندگانه است. این شامل شاخه‌هایی با اندازه هسته‌های مختلف (۱*۱ و ۳*۳ و ۵*۵) و ادغام است که قدرت نمایش شبکه و قابلیت استخراج ویژگی را افزایش می‌دهد.

اهمیت لایه‌های کانولوشن در استخراج ویژگی:

- استخراج ویژگی‌های محلی: لایه‌های کانولوشن از فیلترهای قابل یادگیری برای استخراج ویژگی‌های محلی مانند لبه‌ها، بافت‌ها و الگوها از تصاویر ورودی استفاده می‌کنند.
- نمایش سلسه مراتقی: آنها نمایش های سلسه مراتقی از داده‌های ورودی را می‌گیرند، از ویژگی‌های ساده در لایه‌های اولیه (مانند لبه‌ها) تا ویژگی‌های پیچیده تر و انتزاعی تر در لایه‌های عمیق تر. لایه‌های کانولوشن استخراج ویژگی را با اعمال فیلترهای قابل یادگیری برای داده‌های ورودی انجام می‌دهند. فیلترهای مختلف ویژگی‌های متفاوتی (لبه‌ها، بافت‌ها) را در بر می‌گیرند و نقش مهمی در یادگیری نمایش‌های سلسه مراتقی دارند.
- Translation invariance: لایه‌های کانولوشن باد می‌گیرند که نسبت به جا به جایی‌های کوچک تغییر ناپذیر باشند و به شبکه کمک می‌کنند تا روابط فضایی بین پیکسل‌ها را درک کند، که برای تشخیص و درک اشیا بسیار مهم است.

به طور خلاصه، لایه‌های کانولوشنی نقشی محوری در استخراج ویژگی‌های متمایز از داده‌های ورودی بازی می‌کنند، و شبکه را قادر می‌سازد تا بازنمایی‌هایی را بیاموزد که به تدریج انتزاعی تر و مناسب‌تر برای طبقه‌بندی یا شناسایی مؤثر هستند.

پیاده سازی با استفاده از کتابخانه keras انجام شده است.

هدف این تمرین رسیدن به دقت بالای ۸۰ درصد در آموزش است. برای رسیدن به این هدف از همه روش‌هایی که در درس آموخته اید استفاده کنید.

همان طور که در مدل ۱ و ۲ مشاهده می‌شود با استفاده از ماقول inception، داده افزایی(data augmentation)، استفاده از لایه MaxPooling، لایه کانولوشنی (همراه با Dense و padding و stride) و (کاملاً متصل) توانسته ایم به دقت بالای ۸۰ درصد در داده‌های آموختی برسیم.

همچنین با مراجعه به فایل Q7.ipnb می‌توانید کد پیاده سازی شده را مشاهده کنید. در این نوتبوک می‌توانید با خواندن کامنت‌ها و markdown‌ها به طور کامل بفهمید که چه کاری صورت گرفته است. برای مقایسه نتایج مختلف و پیدا کردن مدل بهتر با استفاده از معماری‌ها و تکنیک چهار مدل مختلف پیاده سازی کرده ایم که در زیر ساختار و نتایج به دست آمده از آنها را می‌توانید ببینید.

توضیح خلاصه کد:

- ابتدا کتابخانه‌های لازم وارد می‌کنیم و سپس مجموعه داده خواسته شده را load می‌کنیم.
- سپس به پیش‌پردازش داده‌ها می‌پردازیم. داده‌های ورودی در محدوده ۰ تا ۱ می‌آوریم و برچسب‌ها را به صورت one-hot کد می‌کنیم.
- ماقول inception را به صورت یک تابع پایتون که از لایه‌های کانولوشنی و ادغام استفاده می‌کند تعریف می‌کنیم.
- شبکه عصبی کانولوشنی را با استفاده از ماقول inception می‌سازیم.
- مدل را compile می‌کنیم و سپس با مجموعه داده‌های آموختی آموزش می‌دهیم.

۶. سپس آن را ارزیابی می کنیم.

مدل ۱:

```
# Define the Inception module
def inception_module(x):
    branch1 = Conv2D(filters=32, kernel_size=(1, 1), activation='relu', strides=(1, 1), padding='same')(x)
    branch2 = Conv2D(filters=32, kernel_size=(1, 1), activation='relu', strides=(1, 1), padding='same')(x)
    branch2 = Conv2D(filters=64, kernel_size=(3, 3), activation='relu', strides=(1, 1), padding='same')(branch2)
    branch3 = Conv2D(filters=64, kernel_size=(1, 1), activation='relu', strides=(1, 1), padding='same')(x)
    branch3 = Conv2D(filters=128, kernel_size=(3, 3), activation='relu', strides=(1, 1), padding='same')(branch3)
    branch4 = MaxPooling2D(pool_size=(3, 3), strides=(1, 1), padding='same')(x)
    branch4 = Conv2D(filters=32, kernel_size=(1, 1), activation='relu', strides=(1, 1), padding='same')(branch4)
    concatenated = tf.concat([branch1, branch2, branch3, branch4], axis=-1)
    return concatenated

# Define the convolutional neural network model
inputs = Input(shape=(32, 32, 3))
x = Conv2D(filters=32, kernel_size=(3, 3), activation='relu', strides=(1, 1), padding='same')(inputs)
x = MaxPooling2D(pool_size=(2, 2), strides=(2, 2), padding='same')(x)
x = inception_module(x)
x = MaxPooling2D(pool_size=(2, 2), strides=(2, 2), padding='same')(x)
x = Flatten()(x)
x = Dense(units=512, activation='relu')(x)
outputs = Dense(units=10, activation='softmax')(x)
model = Model(inputs=inputs, outputs=outputs)
# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
# Data augmentation
datagen = tf.keras.preprocessing.image.ImageDataGenerator(horizontal_flip=True, zoom_range=0.2)
# Train the model
model.fit(datagen.flow(x_train, y_train, batch_size=32), epochs=10, validation_data=(x_test, y_test))
```

دقت پس مدل ۱ بعد از ده مرحله آموزش:

```

Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
178498071/170498071 [=====] - 3s 0us/step
Epoch 1/10
1563/1563 [=====] - 48s 22ms/step - loss: 1.3319 - accuracy: 0.5249 - val_loss: 1.1410 - val_accuracy: 0.5905
Epoch 2/10
1563/1563 [=====] - 38s 24ms/step - loss: 0.9839 - accuracy: 0.6531 - val_loss: 0.9110 - val_accuracy: 0.6854
Epoch 3/10
1563/1563 [=====] - 38s 24ms/step - loss: 0.8472 - accuracy: 0.7037 - val_loss: 0.8792 - val_accuracy: 0.6959
Epoch 4/10
1563/1563 [=====] - 42s 27ms/step - loss: 0.7582 - accuracy: 0.7340 - val_loss: 0.8489 - val_accuracy: 0.7149
Epoch 5/10
1563/1563 [=====] - 37s 24ms/step - loss: 0.6844 - accuracy: 0.7602 - val_loss: 0.7623 - val_accuracy: 0.7403
Epoch 6/10
1563/1563 [=====] - 37s 24ms/step - loss: 0.6222 - accuracy: 0.7807 - val_loss: 0.7747 - val_accuracy: 0.7432
Epoch 7/10
1563/1563 [=====] - 35s 23ms/step - loss: 0.5750 - accuracy: 0.7986 - val_loss: 0.7384 - val_accuracy: 0.7539
Epoch 8/10
1563/1563 [=====] - 40s 25ms/step - loss: 0.5194 - accuracy: 0.8192 - val_loss: 0.7837 - val_accuracy: 0.7552
Epoch 9/10
1563/1563 [=====] - 40s 25ms/step - loss: 0.4864 - accuracy: 0.8294 - val_loss: 0.7087 - val_accuracy: 0.7710
Epoch 10/10
1563/1563 [=====] - 39s 25ms/step - loss: 0.4478 - accuracy: 0.8421 - val_loss: 0.7357 - val_accuracy: 0.7633
<keras.src.callbacks.History at 0x7f6a93fecfa0>

```

همان طور که سوال خواسته بود این مدل توانسته است به دقت بالای ۸۰٪ درصد (۸۴٪ درصد) روی داده های آموزشی برسد و روی داده های تست هم به ۷۷٪ درصد رسیده است.

:۲ مدل

```

def inception_module(x):
    branch_1x1 = Conv2D(16, (1, 1), padding='same', activation='relu')(x)
    branch_3x3 = Conv2D(16, (1, 1), padding='same', activation='relu')(x)
    branch_3x3 = Conv2D(32, (3, 3), padding='same', activation='relu')(branch_3x3)
    branch_5x5 = Conv2D(16, (1, 1), padding='same', activation='relu')(x)
    branch_5x5 = Conv2D(32, (5, 5), padding='same', activation='relu')(branch_5x5)
    branch_pool = MaxPooling2D((3, 3), strides=(1, 1), padding='same')(x)
    branch_pool = Conv2D(16, (1, 1), padding='same', activation='relu')(branch_pool)
    output = concatenate([branch_1x1, branch_3x3, branch_5x5, branch_pool], axis=-1)
    return output

input_layer = Input(shape=(32, 32, 3))
conv1 = Conv2D(32, (3, 3), strides=(1, 1), padding='same',
activation='relu')(input_layer)
conv2 = Conv2D(64, (3, 3), strides=(2, 2), padding='same', activation='relu')(conv1)
inception = inception_module(conv2)
flatten = Flatten()(inception)
dense = Dense(128, activation='relu')(flatten)
output = Dense(10, activation='softmax')(dense)
model = Model(inputs=input_layer, outputs=output)
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
history = model.fit(train_images, train_labels, epochs=20, batch_size=128,
validation_data=(test_images, test_labels))

```

دقت پس مدل ۲ بعد از بیست مرحله آموزش:

```

391/391 [=====] - 5s 13ms/step - loss: 0.1056 - accuracy: 0.9648 - val_loss: 1.6491 - val_accuracy: 0.6849
Epoch 10/20
391/391 [=====] - 5s 13ms/step - loss: 0.0860 - accuracy: 0.9711 - val_loss: 1.7847 - val_accuracy: 0.6861
Epoch 11/20
391/391 [=====] - 5s 13ms/step - loss: 0.0719 - accuracy: 0.9759 - val_loss: 1.8813 - val_accuracy: 0.6736
Epoch 12/20
391/391 [=====] - 5s 14ms/step - loss: 0.0672 - accuracy: 0.9769 - val_loss: 1.9919 - val_accuracy: 0.6736
Epoch 13/20
391/391 [=====] - 5s 12ms/step - loss: 0.0566 - accuracy: 0.9810 - val_loss: 2.1602 - val_accuracy: 0.6732
Epoch 14/20
391/391 [=====] - 5s 13ms/step - loss: 0.0543 - accuracy: 0.9816 - val_loss: 2.2011 - val_accuracy: 0.6834
Epoch 15/20
391/391 [=====] - 5s 14ms/step - loss: 0.0503 - accuracy: 0.9828 - val_loss: 2.3533 - val_accuracy: 0.6727
Epoch 16/20
391/391 [=====] - 5s 13ms/step - loss: 0.0559 - accuracy: 0.9812 - val_loss: 2.3019 - val_accuracy: 0.6752
Epoch 17/20
391/391 [=====] - 5s 14ms/step - loss: 0.0471 - accuracy: 0.9839 - val_loss: 2.5658 - val_accuracy: 0.6738
Epoch 18/20
391/391 [=====] - 5s 13ms/step - loss: 0.0533 - accuracy: 0.9818 - val_loss: 2.4602 - val_accuracy: 0.6789
Epoch 19/20
391/391 [=====] - 5s 12ms/step - loss: 0.0418 - accuracy: 0.9859 - val_loss: 2.4561 - val_accuracy: 0.6667
Epoch 20/20
391/391 [=====] - 5s 13ms/step - loss: 0.0378 - accuracy: 0.9871 - val_loss: 2.5791 - val_accuracy: 0.6695

```

همان طور که سوال خواسته بود این مدل توانسته است به دقت بالای ۸۰ درصد(۹۸ درصد) روی داده های آموزشی برسد و روی داده های تست هم به ۶۷ درصد رسیده است.

مدل :۳

```

def inception_module(prev_layer, filters):
    conv1x1 = Conv2D(filters=filters[0], kernel_size=(1, 1), activation='relu',
padding='same')(prev_layer)
    conv3x3 = Conv2D(filters=filters[1], kernel_size=(3, 3), activation='relu',
padding='same')(prev_layer)
    conv5x5 = Conv2D(filters=filters[2], kernel_size=(5, 5), activation='relu',
padding='same')(prev_layer)
    maxpool = MaxPooling2D(pool_size=(3, 3), strides=(1, 1),
padding='same')(prev_layer)
    pool_conv = Conv2D(filters=filters[3], kernel_size=(1, 1), activation='relu',
padding='same')(maxpool)
    # Concatenate the outputs of different convolutions
    output = concatenate([conv1x1, conv3x3, conv5x5, pool_conv], axis=-1)
    return output
input_layer = Input(shape=(32, 32, 3))
# First Convolutional layer
conv1 = Conv2D(64, (3, 3), activation='relu', padding='same')(input_layer)
conv1 = MaxPooling2D(pool_size=(2, 2))(conv1)
# Apply Inception module
inception1 = inception_module(conv1, [32, 32, 32, 32])
# Additional Convolutional layers
conv2 = Conv2D(128, (3, 3), activation='relu', padding='same')(inception1)
conv2 = MaxPooling2D(pool_size=(2, 2))(conv2)
# Flatten the output for Dense layers
flatten = Flatten()(conv2)
# Dense layers

```

```

dense1 = Dense(256, activation='relu')(flatten)
dense1 = Dropout(0.5)(dense1)
output_layer = Dense(num_classes, activation='softmax')(dense1)
# Define the model
model = Model(inputs=input_layer, outputs=output_layer)
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

# Data augmentation
datagen = ImageDataGenerator(rotation_range=15, width_shift_range=0.1,
                             height_shift_range=0.1, horizontal_flip=True)
datagen.fit(x_train)

# Train the model
batch_size = 64
epochs = 30
history = model.fit(datagen.flow(x_train, y_train, batch_size=batch_size),
                     steps_per_epoch=x_train.shape[0] // batch_size,
                     epochs=epochs,
                     validation_data=(x_test, y_test))

```

دقت پس مدل ۳ بعد از سی مرحله آموزش:

```

781/781 [=====] - 31s 40ms/step - loss: 0.7465 - accuracy: 0.7424 - val_loss: 0.6551 - val_accuracy: 0.7780
Epoch 21/30
781/781 [=====] - 31s 40ms/step - loss: 0.7338 - accuracy: 0.7472 - val_loss: 0.6297 - val_accuracy: 0.7838
Epoch 22/30
781/781 [=====] - 33s 43ms/step - loss: 0.7277 - accuracy: 0.7480 - val_loss: 0.6575 - val_accuracy: 0.7698
Epoch 23/30
781/781 [=====] - 31s 40ms/step - loss: 0.7101 - accuracy: 0.7545 - val_loss: 0.6437 - val_accuracy: 0.7812
Epoch 24/30
781/781 [=====] - 31s 40ms/step - loss: 0.7048 - accuracy: 0.7544 - val_loss: 0.6910 - val_accuracy: 0.7718
Epoch 25/30
781/781 [=====] - 34s 44ms/step - loss: 0.6991 - accuracy: 0.7584 - val_loss: 0.6685 - val_accuracy: 0.7739
Epoch 26/30
781/781 [=====] - 31s 40ms/step - loss: 0.6950 - accuracy: 0.7596 - val_loss: 0.6401 - val_accuracy: 0.7853
Epoch 27/30
781/781 [=====] - 31s 40ms/step - loss: 0.6859 - accuracy: 0.7646 - val_loss: 0.6233 - val_accuracy: 0.7908
Epoch 28/30
781/781 [=====] - 31s 40ms/step - loss: 0.6862 - accuracy: 0.7614 - val_loss: 0.6584 - val_accuracy: 0.7763
Epoch 29/30
781/781 [=====] - 33s 43ms/step - loss: 0.6747 - accuracy: 0.7653 - val_loss: 0.6617 - val_accuracy: 0.7794
Epoch 30/30
781/781 [=====] - 32s 40ms/step - loss: 0.6658 - accuracy: 0.7706 - val_loss: 0.6225 - val_accuracy: 0.7939

```

این مدل توانسته است به دقت ۷۷ درصد روی داده های آموزشی برسد و روی داده های تست هم به ۷۹ درصد(تقریباً ۸۰ درصد رسیده است).

مدل ۴:

```

def inception_module(prev_layer, filters):
    conv1x1 = Conv2D(filters=filters[0], kernel_size=(1, 1), activation='relu',
                     padding='same')(prev_layer)

```

```

        conv3x3 = Conv2D(filters=filters[1], kernel_size=(3, 3), activation='relu',
padding='same')(prev_layer)
        conv5x5 = Conv2D(filters=filters[2], kernel_size=(5, 5), activation='relu',
padding='same')(prev_layer)
        maxpool = MaxPooling2D(pool_size=(3, 3), strides=(1, 1),
padding='same')(prev_layer)
        pool_conv = Conv2D(filters=filters[3], kernel_size=(1, 1), activation='relu',
padding='same')(maxpool)
        # Concatenate the outputs of different convolutions
        output = concatenate([conv1x1, conv3x3, conv5x5, pool_conv], axis=-1)
        return output
# Define the regularization parameter
regularization_rate = 0.001 # You can adjust this value
input_layer = Input(shape=(32, 32, 3))
# First Convolutional layer with L2 regularization
conv1 = Conv2D(64, (3, 3), activation='relu', padding='same',
kernel_regularizer=l2(regularization_rate))(input_layer)
conv1 = MaxPooling2D(pool_size=(2, 2))(conv1)
# Apply Inception module
inception1 = inception_module(conv1, [32, 32, 32, 32])
# Additional Convolutional layers with L2 regularization
conv2 = Conv2D(128, (3, 3), activation='relu', padding='same',
kernel_regularizer=l2(regularization_rate))(inception1)
conv2 = MaxPooling2D(pool_size=(2, 2))(conv2)
# Flatten the output for Dense layers
flatten = Flatten()(conv2)
# Dense layers with L2 regularization
dense1 = Dense(256, activation='relu',
kernel_regularizer=l2(regularization_rate))(flatten)
dense1 = Dropout(0.5)(dense1)
output_layer = Dense(num_classes, activation='softmax')(dense1)
# Define the model
model = Model(inputs=input_layer, outputs=output_layer)
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
# Data augmentation
datagen = ImageDataGenerator(rotation_range=15, width_shift_range=0.1,
height_shift_range=0.1, horizontal_flip=True)
datagen.fit(x_train)
# Train the model
batch_size = 64
epochs = 30
history = model.fit(datagen.flow(x_train, y_train, batch_size=batch_size),
                     steps_per_epoch=x_train.shape[0] // batch_size,

```

```
epochs=epochs,  
validation_data=(x_test, y_test))
```

دقت پس مدل ۴ بعد از سی مرحله آموزش:

```
781/781 [=====] - 33s 42ms/step - loss: 1.1342 - accuracy: 0.6710 - val_loss: 0.9910 - val_accuracy: 0.7178  
Epoch 20/30  
781/781 [=====] - 34s 44ms/step - loss: 1.1282 - accuracy: 0.6732 - val_loss: 0.9715 - val_accuracy: 0.7313  
Epoch 21/30  
781/781 [=====] - 33s 42ms/step - loss: 1.1098 - accuracy: 0.6817 - val_loss: 0.9378 - val_accuracy: 0.7430  
Epoch 22/30  
781/781 [=====] - 36s 46ms/step - loss: 1.1131 - accuracy: 0.6791 - val_loss: 1.0264 - val_accuracy: 0.7201  
Epoch 23/30  
781/781 [=====] - 33s 42ms/step - loss: 1.1095 - accuracy: 0.6846 - val_loss: 1.0686 - val_accuracy: 0.7032  
Epoch 24/30  
781/781 [=====] - 33s 42ms/step - loss: 1.1070 - accuracy: 0.6845 - val_loss: 0.9518 - val_accuracy: 0.7361  
Epoch 25/30  
781/781 [=====] - 34s 43ms/step - loss: 1.1035 - accuracy: 0.6830 - val_loss: 0.9649 - val_accuracy: 0.7347  
Epoch 26/30  
781/781 [=====] - 35s 45ms/step - loss: 1.0957 - accuracy: 0.6879 - val_loss: 0.9261 - val_accuracy: 0.7451  
Epoch 27/30  
781/781 [=====] - 33s 42ms/step - loss: 1.0822 - accuracy: 0.6925 - val_loss: 0.9843 - val_accuracy: 0.7204  
Epoch 28/30  
781/781 [=====] - 33s 42ms/step - loss: 1.0821 - accuracy: 0.6952 - val_loss: 0.9782 - val_accuracy: 0.7323  
Epoch 29/30  
781/781 [=====] - 34s 44ms/step - loss: 1.0803 - accuracy: 0.6909 - val_loss: 0.9958 - val_accuracy: 0.7257  
Epoch 30/30  
781/781 [=====] - 33s 42ms/step - loss: 1.0773 - accuracy: 0.6961 - val_loss: 0.9490 - val_accuracy: 0.7409
```

این مدل توانسته است به دقت ۶۹ درصد روی داده های آموزشی برسد و روی داده های تست هم به ۷۴ درصد رسیده است.

مراجع:

<https://chat.openai.com/>

<https://bard.google.com/>

<https://claude.ai/chats>

پایان