تمرین سری چهارم

عماد الدین رستیان

1 (الف) با فرض stride=1 و تعداد فیلتر F

$$\begin{cases} W_I = \frac{W-k}{1}+1 \\ H_I = \frac{H-k}{1}+1 \\ D_I = F \end{cases}$$

تعداد وزن ها : $F \times (D \times K \times K)$

تعداد اتصالات $(W_I \times H_I) \times F \times (D \times K \times K)$

برای لایه متراکم ابتدا باید تصویر به صورت یک بردار تک بعدی تبدیل شود که طول آن $W \times H \times D$ است پس تعداد وزن ها: $W \times H \times D \times F$ که همان تعداد اتصالات نیز هست

همانطور که مشاهده می‌شود تعداد وزن ها در شبکه پیچشی کمتر از تعداد وزن ها در شبکه متراکم است بنابراین برای آموزش شبکه متراکم به تعداد داده بسیار بیشتری نیاز دارد (برای جلوگیری از بیش برازش).

ب) در شبکه های متراکم:

$$x = \underset{\text{ماتریس وزن ها}}{W}u + \underset{\text{بردار بایاس}}{b} \xrightarrow{BN} \hat{x} = \frac{x-\mu_B}{\sqrt{\sigma_B^2+\epsilon}} \xrightarrow{\text{shift \& scale}} y = \gamma\hat{x} + \beta$$

بردار ورودی های نورون

در این شبکه نرمال سازی به هر نورون مستقلا اعمال می‌شود.

در شبکه های کانولوشنی:

$$x = \underset{\text{Kernel}}{W} * \underset{\text{جمع وزنی}}{u} + b \xrightarrow{BN} \hat{x}_{i,j} = \frac{x_{i,j}-\mu_B}{\sqrt{\sigma_B^2+\epsilon}} \xrightarrow[\text{\& scale}]{\text{shift}} y_{i,j} = \gamma\hat{x}_{i,j} + \beta$$

در این شبکه نرمال سازی بر روی feature map اعمال می‌شود.

در صورت عدم استفاده از BN نبود بایاس ممکن است شبکه را دچار مشکل کند زیرا بایاس با تنظیم offset خروجی باعث یادگیری بهتر داده ها می‌شود. در صورت استفاده از BN نبود بایاس تاثیری در شبکه نمی‌گذارد زیرا BN با shift & scale همان کار بایاس را انجام می‌دهد.

به دلیل BN ضرب شدن وزن ها در α تاثیری ندارد زیرا در نهایت BN اثرش را خنثی می‌کند. پس به وزن ها، ضرب α در ورودی تاثیری در عملکرد شبکه نخواهد داشت.

در پردازش تصاویر پزشکی، به دلیل رزولوشن بالای آن‌ها و به نتیجه آن حجم فضای بالا در حافظه، معمولا اندازه batch ها کوچکتر انتخاب می‌شود. batch با سایز کوچک معمولا باعث ناپایداری و تخمین های نویزی می‌شوند. راه حل Group Normalization ؛ نرمال‌سازی را در گروه های تقسیم شده توسط کانال ها انجام می‌شود.

(ج)

1st layer: 256×256 → 128×128

2nd layer: 128×128 → 64×64

3rd layer: 64×64 → 32×32

4th layer: 32×32 → 16×16

5th layer: 16×16 → 8×8 → 64 pixels

تعداد پارامترها = $3 \times 3 \times 64 \times 128 + 128 = 73856$

(د) Up-Convolutional : این روش با بازسازی یک تصویر از ویژگی های استخراج شده است. معمولا در نشان دهد که در هر بازنمایی از ویژگی ها به اطلاعاتی حفظ می‌شود. این روش ترکیبی از convolution و upsampling است که نشان می‌دهد در لایه کانولوشن اول ویژگی های سطح پایین مانند لبه‌ها و بافت‌ها در لایه های میانی مانند لایه کانولوشن سوم ویژگی ها و الگوهای محلی دریافت می‌شود مانند شکل و بخش های مختلف اجسام. در لایه های بالاتر مانند FC7 و FC8 بر ویژگی های سطح بالا متمرکز است. این روش با بازسازی و حفظ رنگ تصویر نشان می‌دهد که لایه های کانولوشن اطلاعات بسیار زیادی را ذخیره می‌کنند.

Deconvolutional : این شبکه بخش های از تصویر را مشخص می‌کند که بیشترین تاثیر را در فعال‌سازی یک سری نورون ها یا ویژگی های خاص دارند. در واقع این شبکه عملیات مستقیم CNN را معکوس می‌کند:

1- Unpooling 2- Deconvolution 3- Filtering

یک گروه از نورون ها انتخاب می‌شود و فعال‌سازی آن‌ها توسط backpropagation منجر به مورد سازی قسمت های مرتبط با آن گروه نورون ها می‌شود.

دو روش معرفی شده می‌توانند ابزاری برای بررسی جعبه سیاه شبکه های کانولوشن باشند. Up convolutional در تجسم آنچه شبکه در کل ورودی حفظ می‌کند و deconvolutional در تجسم پاره‌ای از نورون ها به تغییر پذیری کمک می‌کند.

CamScanner

2

$$\frac{\partial L}{\partial w_j} = \sum_i \frac{\partial L}{\partial z_i} \cdot \frac{\partial z_i}{\partial w_j}$$

$$\Rightarrow \frac{\partial L}{\partial w_j} = \sum_i \frac{\partial L}{\partial z_i} \cdot x_{i+j} = \left(\frac{\partial L}{z} * x\right)_j$$

$$z_i = \sum_{j=0}^{k-1} w_j x_{i+j} \rightarrow \frac{\partial z_i}{\partial w_j} = x_{i+j}$$

3

$$P^{(3)}_{i,j,k} = \frac{1}{4} \sum_{m=0}^{1} \sum_{n=0}^{1} P_{2i+m,\,2j+n,\,k} \quad , \quad \frac{\partial L}{\partial P^{(3)}} = \frac{\partial L}{\partial z}$$

(ت)

$$\frac{\partial L}{\partial P^{(2)}_{p,q,k}} = \sum_{i,j} \frac{\partial L}{\partial P^{(3)}_{i,j,k}} \cdot \underbrace{\frac{\partial P^{(3)}_{i,j,k}}{\partial P_{p,q,k}}}_{1/4} = \frac{1}{4} \sum_{i,j} \frac{\partial L}{\partial P^{(3)}_{i,j,k}}$$

$$\frac{\partial L}{\partial P^{(1)}_{i,j,l}} = \sum_{p,q,k} \frac{\partial L}{\partial P^{(2)}} \cdot \frac{\partial P^{(2)}}{\partial P^{(1)}}$$

$$\Rightarrow \frac{\partial L}{\partial P^{(1)}_{m,n,l}} = \frac{1}{4} \sum_k \sum_{i,j} \frac{\partial L}{\partial z} \cdot W^{(2)}_{k,l}$$

$$P^{(2)}_{i,j,k} = \sum_{m,n,l} W^{(2)}_{m,n,l,k} \cdot P^{(1)}_{i+m,\,j+n,\,l}$$

(ب)

$$\frac{\partial L}{\partial W^{(1)}_{1,1,k}} = \sum_{i,j} \frac{\partial L}{\partial P^{(1)}_{i,j,k}} \cdot \underbrace{\frac{\partial P^{(1)}_{i,j,k}}{\partial W^{(1)}_{1,1,k}}}_{X} = \frac{1}{4} \sum_k \sum_{i,k} \frac{\partial L}{\partial z} X_{k,1}$$

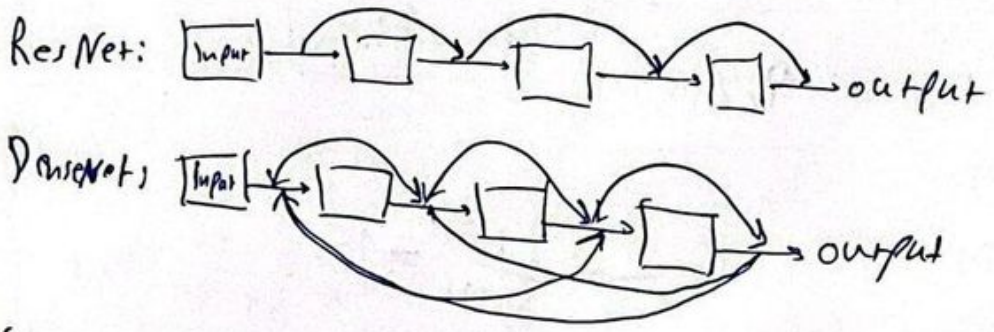
$$\frac{\partial P^{(1)}}{\partial W^{(1)}} = X \longrightarrow$$

(ج)

$$\frac{\partial L}{\partial W^{(2)}} = \sum \frac{\partial L}{\partial P^{(2)}} \cdot \underbrace{\frac{\partial P^{(2)}}{\partial W^{(2)}}}_{P^{(1)}} = \frac{1}{4} \sum_{m,n} \frac{\partial L}{\partial z} \cdot P^{(1)}_{c,m+i,n+j}$$

۴) آ) DenseNet تمام لایه‌ها را به لایه‌های قبلی متصل می‌کند. این اتصال باعث جلوگیری از گرادیان ناپدید شونده می‌شود. این شبکه برای ترکیب ورودی‌ها از concatination استفاده می‌کند که باعث می‌شود مدل فشرده و روند گرادیان بهینه شود که خطر overfitting را کاهش می‌دهد.

ResNet مشکل گرادیان ناپدید شونده را با اتصالات Residual حل می‌کند. این اتصالات به لایه‌ها اجازه می‌دهد تا ورودی خود را به لایه‌های عمیق‌تر ارسال کنند. این عملیات از جمع خروجی لایه عمیق‌تر با ورودی لایه‌های قبلی انجام می‌شود و در صورت صفر شدن گرادیان اتصال residual با جمع کردن گرادیان ورودی مانع از صفر شدن کل گرادیان می‌شود.



همانطور که گفته شد تفاوت اصلی دو مدل نحوه اتصالات و نحوه ترکیب کردن آن‌هاست (جمع، concatination)

به دلیل وجود اتصالات مستقیم از loss function به تمام لایه‌ها، بعلاوه ورودی اصلی، باعث می‌شود تا گرادیان به راحتی در شبکه جریان داشته باشد بدون آن که ناپدید شود

مزیت‌های محاسباتی DenseNet:

1- عدم نیاز به یادگیری مجدد تعداد زیادی ویژگی به دلیل استفاده مجدد خروجی لایه‌های اول

2- به دلیل یادگیری تجمعی این مدل، DenseNet نیازی به تعداد زیاد فیلتر ندارد.

3- اتصالات متراکم این مدل نوعی Regularization محسوب می‌شود که overfitting را کاهش می‌دهد

K=24

Third layer → 32+3K = 104

۴ (ب)

تعداد عملیات بدون conv 1×1 $= 28 \times 28 \times 32 \times 5 \times 5 \times 192$

$= 120422400$

تعداد عملیات با conv 1×1 $= 28 \times 28 \times 16 \times 1 \times 1 \times 192 + 28 \times 28 \times 32 \times 5 \times 5 \times 16$

$= 12443648$

درصد کاهش عملیات $= \dfrac{120422400 - 12443648}{120422400} \simeq 89.7\%$

ترکیب فیلترهای مختلف به مدل اجازه می‌دهد تا الگوهای محلی وسیع‌تری را بهتر یاد بگیرد.

مسیر اول:

64 Filters (1×1)

padding = 0

stride = 1

خروجی $= 28 \times 28 \times 64$

تعداد عملیات $= 28 \times 28 \times 64 \times (1 \times 1 \times 192)$

$= 9830400$

مسیر دوم:

96 (1×1) + 128 (3×3);

padding = 0 padding = 1

stride = 1 stride = 1

خروجی $= 28 \times 28 \times 128$

تعداد عملیات $= 28 \times 28 \times 96 \times 192 + 28 \times 28 \times 128 \times 3 \times 3 \times 96$

$= 42004224$

مسیر سوم:

16 (1×1) + 32 (5×5)

Padding = 0 Padding = 2

stride = 1 stride = 1

خروجی $= 28 \times 28 \times 32$

تعداد عملیات $= 28 \times 28 \times 16 \times 192 + 28 \times 28 \times 32 \times 5 \times 5 \times 16$

$= 12779526$

مسیر چهارم:

Max pool + 32 (1×1)

P = 1 P = 0

S = 1 S = 1

خروجی $= 28 \times 28 \times 32$

تعداد عملیات $= 0 + 28 \times 28 \times 32 \times 1 \times 1 \times 192 = 4915200$

تعداد کل عملیات = 67529344

دلیل این معماری به گرادیان ناپدید شدن مقاوم است زیرا :

1- فیلتر 1x1 مانند لایه bottleneck عمل می‌کند که باعث کاهش ابعاد ویژگی‌ها می‌شود و مسیر گرادیان را هموارتر می‌کند

2- Auxiliary classifier های موجود در این معماری با افزودن loss های اضافه مانند میانبری برای گرادیان عمل می‌کنند که باعث می‌شود لایه های اول سیگنال گرادیان قوی‌تری دریافت کنند

3- وجود مسیرهای موازی باعث می‌شود اگر گرادیان در مسیری ناپدید شود در مسیرهای دیگر این اثر جبران شود.

Auxiliary classifier ها classifier هایی هستند که در لایه های عمیق شبکه قرار دارند. این classifier ها با استفاده از ویژگی های استخراج شده از لایه های میانی شبکه یک خروجی تولید می‌کنند. این classifier ها فقط در زمان آموزش کار می‌کنند و در زمان Inference نادیده گرفته می‌شوند.

محدودیت های این مدل : 1- پیچیدگی طراحی و معماری

2- از نظر فضای حافظه سنگین

3- امکان هزینه اضافی در Auxiliary classifier ها

4- عدم انعطاف پذیری در اندازه فیلتر ها

5- عدم وجود skip connection

(ج) تفاوت اصلی CNN معمولی با Deformable در sampling grid است. در CNNs کلاسیک sampling grid ثابت و معین است ولی در deformable با یک offset augmented می‌شود تا بتواند با ورودی خود انطباق دهد. learned offset به صورت پویا و براساس نگاشت های قبلی تعیین می‌شود این باعث می‌شود که sampling grid غیرمعین ولی انعطاف پذیر باشد.

شبکه های پیچشی deformable به دلیل adaptive sampling و درون یابی دو خطی می‌توانند در برابر تبدیل های هندسی منعطف باشند

offset از پارامتر ها و ویژگی های قبلی با تبدیل convolution محاسبه می‌شود.

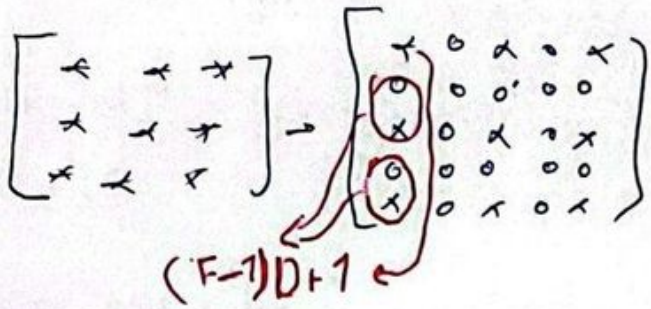
Q (الف) با توجه به شکل

$D=1 \longrightarrow$ Effective field $= F \times F = 3\times 3$

$D=2 \longrightarrow$ Effective field $= 5\times 5$

$D=3 \longrightarrow$ " " $= 7\times 7$

General: $D \longrightarrow (F-1)\times D+1$



Input: $M \times N$

$$\text{output} = \left(M-((F-1)D+1)+1\right)\times\left(N-((F-1)D+1)+1\right)$$

$$= (M-FD+D)\times(N-FD+D)$$

ب)

$$K \otimes A = \begin{bmatrix} k_{11}A & \cdots & k_{1F}A \\ \vdots & \ddots & \vdots \\ k_{F1}A & \cdots & k_{FF}A \end{bmatrix}$$

$$D=2 \rightarrow \begin{bmatrix} k_{11} & 0 & k_{12} & 0 & k_{1F} \\ 0 & 0 & 0 & 0 & 0 \\ k_{21} & 0 & k_{22} & 0 & \vdots \\ 0 & 0 & 0 & 0 & 0 \\ k_{F1} & 0 & \cdots & 0 & k_{FF} \end{bmatrix} \Longrightarrow A = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$$

$$\rightarrow A_{D\times D} = \begin{cases} a_{11}=1 \\ a_{ij}=0 \quad i\neq 1, j\neq 1 \end{cases}$$

$\longrightarrow$ last row should be dropped (All zeros)

ج)

Layer 1: $F_1 = w-1$, $D_1 = d-1$ $\longrightarrow$ Receptive Field: $D_1(F_1-1)+1 = (d-1)(w-2)+1$

Layer 2: $F_2 = w$, $D_2 = d$ $\rightarrow$ Receptive Field: $d(w-1)+1 + R_1 - 1 = d(w-1)+1+(d-1)(w-2)$

Layer 3: $F_2 = w+1$, $D_2 = d+1$ $\longrightarrow$ Receptive Field: $(d+1)w+1 + R_2 - 1 = w(d+1)+d(w-1)+(d-1)(w-2)+1$

$$= (3w-3)d+3$$

CamScanner

۷) Mask convolution : یک نوع از کانولوشن که یک mask به کرنل اعمال می‌شود تا وزن های خاصی را صفر کند تا تاثیر برخی نورون های ورودی را از بین ببرد. کاربرد این کانولوشن معمولا در پردازش داده های sequential است. برای مثال mask convolution تضمین می‌کند که پیکسل خروجی صرفا به پیکسل های قبلی وابسته است و نه به پیکسل های بعدی. این روش در پیش‌بینی سری زمانی نیز به کار برده است.

از محدودیت های این روش می‌توان به موارد زیر اشاره کرد:

1- محدودیت های سختگیرانه: ماسک ها ممکن است در برخی سناریوها بسیار سختگیرانه عمل کنند به خصوص در مواردی که وابستگی ها رنج وسیعی دارند.

2- گرادیان تنک: به دلیل masking جریان گرادیان تنک می‌شود که فرایند بهینه‌سازی را دشوار می‌کند.

3- کاهش Receptive Field: عملیات masking باعث کاهش Receptive field می‌شود

با استفاده از کانولوشن گسترش یافته در mask convolution می‌توان محدودیت های بالا را کاهش داد. با گسترش receptive field عملیات masking به صورت بهینه‌تر و کنترل شده‌تر انجام می‌شود که باعث بهبود جریان گرادیان و کارایی مدل می‌شود.

۶) آ) با فرض بزرگترین مقدار تکین برای W_k داریم:

$$\left\| \prod_{i=1}^{L} W_k \right\| \leq (\sigma_{max})^n \quad , \quad n = L-i$$

از آنجا که $\sigma_{max} < 1$ با بزرگ بودن n به سمت بی‌نهایت σ_{max}^n به صفر میل می‌کند که با توجه به رابطه بازگشتی در صورت سوال $\delta_i \to 0$ که باعث گرادیان ناپدید شونده می‌شود

$$\sigma_{max} = 0.9 \, , \, L = 100$$

$$\delta_L = M \rightarrow |\delta_i| \leq M.(0.9)^{L-i} \xrightarrow{i=0} |\delta_0| \leq M.(0.9)^{100}$$

$$\rightarrow |\delta_0| \leq 2.6 \times 10^{-5} \times M$$

(ب) طبق Kaiming داریم:

$$Var[W] = \frac{2}{n_l}$$

$$n_l = \text{Input channels} \times \text{Kernel size} = 3 \times 8 \times 8 = 192$$

$$Var[W] = \frac{2}{192} = 0.0104 \rightarrow \sigma = \sqrt{Var} = 0.102$$

$$\text{Normalization} \rightarrow \mu = 0 \rightarrow W_l \sim N(0, (0.102)^2)$$

(ج) در Sigmoid برای داده‌های بسیار بزرگ یا بسیار کوچک به مقدارهای 1 و 0 اشباع می‌شود که باعث به وجود آمدن گرادیان صفر می‌شود. تابع ReLU در مقادیر مثبت همواره گرادیان ثابت +1 را دارد و برای مقادیر منفی صفر می‌شود که لزوماً به گرادیان صفر منجر نمی‌شود زیرا همواره نورون‌های مثبت وجود دارند.

(د) مشکل نورون‌های مرده وقتی رخ می‌دهد که مقادیر نورون‌ها منفی باشد. خروجی ReLU برای نورون‌های منفی صفر است که باعث گرادیان صفر هم می‌شود. راه حل:

1- PReLU: $f(x) = \begin{cases} x & x > 0 \\ \alpha x & x \leq 0 \end{cases} \rightarrow f'(x) = \begin{cases} 1 & x > 0 \\ \alpha & x \leq 0 \end{cases}$

مزایا: α یک پارامتر قابل یادگیری است، حل مشکل نورون‌های مرده

معایب: افزودن پارامتر، بالا رفتن پیچیدگی مدل و به نتیجه آن افزایش ریسک overfitting

2- Leaky ReLU: $f(x) = \begin{cases} x & x > 0 \\ \alpha x & x \leq 0 \end{cases} \rightarrow f'(x) = \begin{cases} 1 & x > 0 \\ \alpha & x \leq 0 \end{cases}$

α = a small constant

مزایا: عدم افزودن پارامتر، جلوگیری از مشکل نورون‌های مرده

معایب: α تعیین شده ممکن است بهینه نباشد، عدم انعطاف‌پذیری

CamScanner

3- ELU: $f(x) = \begin{cases} x & x>0 \\ \alpha(e^{x}-1) & x\le 0 \end{cases} \rightarrow f'(x) = \begin{cases} 1 & x>0 \\ \alpha e^{x} & x\le 0 \end{cases}$

α یک هایپرپارامتر است که معمولا به $\alpha=1$ تنظیم می‌شود.

مزایا: عدم وجود شکستگی در نقطه صفر (smooth)، بهبود جریان گرادیان، مرکزی شده به صفر

معایب: هزینه محاسباتی به دلیل e^{x}، تنظیم هایپرپارامتر α برای یادگیری بهینه

(5

$$y_{block} = F(x) + x$$

$$x_1 = F(x_0) + x_0$$

$$x_2 = F(x_1) + x_1 = F(F(x_0) + x_0) + F(x_0) + x_0$$

$$\frac{\partial x_2}{\partial x_1} = 1 + \frac{\partial F(x_1)}{\partial x_1} \xrightarrow{\frac{\partial F(x_1)}{\partial x_1} \to 0} \frac{\partial x_2}{\partial x_1} = 1 \neq 0$$

به دلیل سیر اضافی گرادیان صفر نمی‌شود. به جای یادگیری نگاشت کامل $H(x)$ هر بلوک صرف نگاشت Residual را یاد می‌گیرد $F(x) = H(x) - x$ که برای بهینه‌سازی بهتر است

در شبکه‌های نرمال:

$$x_1 = F(x_0)$$

$$x_2 = F(x_1)$$

$$\frac{\partial x_2}{\partial x_0} = \frac{\partial F(x_2)}{\partial x_1} \frac{\partial F(x_1)}{\partial x_0}$$

اگر یکی مشتق‌ها کوچک شود (به صفر میل کند) باعث صفر شدن کل گرادیان و ناپدیدی آن می‌شود