مسئله ۱ – چون تصویر RGB است لذا ۳ کانال داریم یعنی سایز تصویر ورودی ۳*۳۲*۳۲ است.

قسمت آ) میدانیم که روابط زیر را داریم :

$$Conv_{output} = \left(\frac{input + 2 * padding - kernel}{stride} + 1\right) * kernel\ Channel\ output$$

$$Pooling_{output} = \left(\frac{input - window\ size}{stride} + 1\right) * input\ depth\ size$$

$$conv_{parameter} = (kernelSize * inputChannel + bias) * convOutputChannel$$

لذا :

$$conv1_{output} = \left(\frac{32 + 0 - 5}{1} + 1\right) * 64 = 28 * 28 * 64$$

$$conv1_{parameter} = (5 * 5 * 3 \ + 1) * 64 = 4864$$

$$conv2_{output} = \left(\frac{28 + 0 - 3}{1} + 1\right) * 64 = 26 * 26 * 64$$

$$conv2_{parameter} = (3 * 3 * 64 + 1) * 64 = 36928$$

$$MaxPool_{output} = \left(\frac{26 - 3}{1} + 1\right) * 64 = 24 * 24 * 64$$

$$maxPool_{parameter} = No\ parameters$$

قسمت ب) میدانیم که :

$$receptiveField_k = 1 + \sum_{j=1}^{k} (F_j - 1) \prod_{i=0}^{j-1} (S_i + d_i - 1)$$

حالت اول – a=1 , b=1 , c=1 , d=1 :

$$receptiveField_3 = 1 + [(5-1)(1) + (3-1)(1+1-1) + (3-1)(1)] = 1 + 4 + 2 + 2 = 9$$

حالت دوم – a=1 , b=1 , c=2 , d=1 :

$$receptiveField_3 = 1 + [(5-1)(1) + (3-1)(2+1-1) + (3-1)(1)] = 1 + 4 + 4 + 2 = 11$$

حالت سوم – a=1 , b=1 , c=2 , d=2 :

$$receptiveField_3 = 1 + [(5-1)(1) + (3-1)(2+2-1) + (3-1)(1)] = 1 + 4 + 4 + 2 = 13$$

لذا میبینیم که با افزایش مقدار dilation و stride یک لایه، receptive field آن لایه افزایش پیدا میکند.دقیق تر بررسی کنیم، با اضافه کردن یک واحد به stride یا dilation ، ما به اندازه یکی کمتر از سایز کرنل، به receptive field اضافه میشود. با این رابطه بالعکسش هم درسته یعنی با کاهش آن ها، همان مقدار از receptive field کم میشود.

<u>دلایل استفاده از dilation :</u> افزایش receptive field بدون بوجود آمدن هزینه در resolution و coverage یعنی با افزایش خطی تعداد پارامتر ها(افزایش خیلی کم حافظه مصرفی و پیچیدگی زمانی) میتوانیم بوسیله کنترل dilation و اضافه کردن این نوع لایه ها، receptive field را به صورت توانی افزایش دهیم.زیرا با داشتن تعداد پارامتر وقتی داریم از convolution معمولی استفاده میکنیم، میتوانیم در dialated convolution منطقه گسترده تری از تصویر را پوشش دهیم.همچنین در dialated convolution با اضافه کردن صفر به kernel، کمک میکنیم که resolution سریع تر کاهش یابد.همچنین از overfitting هم بهتر جلوگیری میشود.

<u>دلایل استفاده از stride:</u> با افزایش stride در لایه های کانولوشن، میتوانیم کاهش بعد(رزولوشن) داشته باشیم و میتوانیم با اینکار حتی از استفاده کردن از لایه های pooling زیاد بپرهیزیم.stride پایین باعث overlap و در نتیجه بزرگ بودن خروجی لایه شود.لذا افزایش stride هم باعث کاهش حافظه مصرفی و پیچیدگی زمانی میشود. با کم شدن حجم خروجی لایه، نتیجه میگیریم تعداد پارامتر ها نیز کم شده لذا مانند حالت قبل، از overfitting هم جلوگیری میشود.

قسمت ج)

معایب: چون در کل pooling نوعی انتخاب کاندید است، لذا loss دارد و کل اطلاعات را حفظ نمیکند – sharp شدن خروجی(بر خلاف average pooling) – از بین بردن localization

مزایا: کاهش واریانس – کاهش بعد خروجی – کاهش تعداد پارامترها – کاهش پیچیدگی محاسباتی – استخراج مهم ترین feature ها در یک همسایگی(بر خلاف average pooling) – blur نشدن خروجی(بر خلاف average pooling)

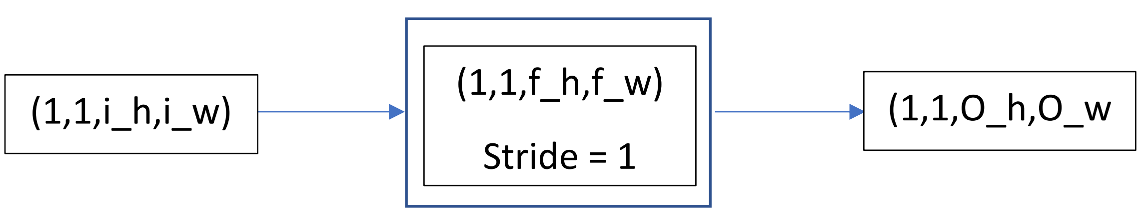
قسمت د) ابتدا در رابطه با depthwise seperable conv صحبت میکنیم.این نوع کانولوشن، یک کانولوشن مکانی(spatial) است که روی هر کانال ورودی(عمق) به صورت مستقل عمل میکند.این نوع کانولوشن، کرنل را به ۲ کرنل کوچک تر تبدیل میکند و که این دو وظیفه دو نوع کانولوشن مختلف دارند: depthwise conv & pointwise conv

ابتدا روی تصویر depthwise اعمال شده و سپس خروجی آن را به pointwise میدهیم(برای افزایش عمق خروجی depthwise).با استفاده از depthwise seperable conv ما میتوانیم تا حد زیادی از تعداد ضرب های کانولوشن عادی بکاهیم.لذا پیچیدگی محاسباتی را پایین بیاوریم.

حال در رابطه با grouped conv صحبت میکنیم. ایده این روش این است که با تقسیم کردن filter ها به یک سری گروه، و اعمال کانولوشن روی تصویر به صورت موازی، از پیچیدگی محاسباتی کانولوشن معمولی بکاهیم و تعداد بیشتری feature یاد بگیریم.این نوع کانولوشن ابتدا در AlexNet دیده شد.همچنین با استفاده از این روش میتوان یادگیری مدل را روی GPU های ضعیف تر اما به صورت موازی انجام داد.

پس با مقایسه این دو روش متوجه میشویم depthwise conv نوعی تکه تکه کردن یک kernel است اما grouped conv نوعی افراز kernel های مختلف و موازی سازی یادگیری است لذا دو رویکرد کاملا متفاوت برای یک هدف یکسان(کاهش پیچیدگی محاسباتی) هستند.

مسئله ۲ –



$$O = \begin{bmatrix} o_{(1,1)} & \cdots & x_{(1,o_w)} \\ \vdots & \ddots & \vdots \\ o_{(o_h,1)} & \cdots & x_{(o_h,o_w)} \end{bmatrix} \quad ,W = \begin{bmatrix} w_{(1,1)} & \cdots & w_{(1,f_w)} \\ \vdots & \ddots & \vdots \\ w_{(f_h,1)} & \cdots & w_{(i_h,f_w)} \end{bmatrix} \quad ,X = \begin{bmatrix} x_{(1,1)} & \cdots & x_{(1,i_w)} \\ \vdots & \ddots & \vdots \\ x_{(i_h,1)} & \cdots & x_{(i_h,i_w)} \end{bmatrix}$$

قسمت آ) کانولوشن همان cross-correlation است وقتی که فیلتر هم به صورت افقی هم عموی، flip شده باشد یعنی برای ماتریس flip شده w داریم:

$$w_{flipped} = \begin{pmatrix} w_{f_h,f_w} & \cdots & w_{f_h,1} \\ \vdots & \ddots & \vdots \\ w_{(1,f_w)} & \cdots & w_{1,1} \end{pmatrix}$$

حال باید *cross-correlation* بین $w_{flipped}$ و X را بدست آوریم:

$$O_{i,j} = \left(X \otimes W_{flipped}\right)_{i,j} = \sum_{m=1}^{f_h} \sum_{n=1}^{f_w} X(i+m-1, j+n-1) W_{flipped}(m,n)$$

$$= \begin{pmatrix} \sum_{m=1}^{f_h} \sum_{n=1}^{f_w} X(m,n) W_{flipped}(m,n) & \cdots & \sum_{m=1}^{f_h} \sum_{n=1}^{f_w} X(m, n+o_w-1) W_{flipped}(m,n) \\ \vdots & \ddots & \vdots \\ \sum_{m=1}^{f_h} \sum_{n=1}^{f_w} X(m+o_h-1, n) W_{flipped}(m,n) & \cdots & \sum_{m=1}^{f_h} \sum_{n=1}^{f_w} X(m+o_h-1, n+o_w-1) W_{flipped}(m,n) \end{pmatrix}$$

قسمت ب) طبق مفروض سوال، ما مشتق تابع هزینه یعنی L را نسبت به O یعنی خروجی داریم.با نامگذاری این مفروض داریم:

$$We\ have\ \delta_{i,j} = \frac{\partial L}{\partial O_{i,j}}$$

هم چنین میدانیم سایز ماتریس خروجی برابر است با: $(i_h - f_h + 1) * (i_w - f_w + 1)$

لذا طبق قاعده زنجیره ای داریم:

$$\frac{\partial L}{\partial w_{m\prime,n\prime}} = \sum_{i=1}^{i_h-f_h}\sum_{j=1}^{i_w-f_w}\frac{\partial L}{\partial O_{i,j}}\frac{\partial O_{i,j}}{\partial w_{m\prime,n\prime}}$$

$$= \sum_{i=1}^{i_h-f_h}\sum_{j=1}^{i_w-f_w}\delta_{i,j}\frac{\partial O_{i,j}}{\partial w_{m\prime,n\prime}}$$

از طرفی میدانیم که خروجی برابر است با:

$$O_{i,j} = \sum_{m=1}^{f_h}\sum_{n=1}^{f_w}X(i+m-1,j+n-1)W_{flipped}(m,n)$$

لذا:

$$\frac{\partial O_{i,j}}{\partial w_{m\prime,n\prime}} = \frac{\partial}{\partial w_{m\prime,n\prime}}\left(X(i,j)W_{flipped}(1{,}1) + \cdots + X(i+m'-1,j+n'-1)W_{flipped}(m',n') + \cdots\right)$$

$$= \frac{\partial}{\partial w_{m',n'}}\left(X(i+m'-1,j+n'-1)W_{flipped}(m',n')\right) = X(i+m'-1,j+n'-1)$$

پس داریم:

$$\frac{\partial L}{\partial w_{m\prime,n\prime}} = \sum_{i=1}^{i_h-f_h}\sum_{j=1}^{i_w-f_w}\delta_{i,j}\frac{\partial O_{i,j}}{\partial w_{i+m',j+n\prime}} = \delta_{i,j}\otimes x_{m',n\prime}$$

برای محاسبه خواسته دیگر سوال، مشابهتا داریم:

$$\frac{\partial L}{\partial x_{m\prime,n\prime}} = \sum_{i=1}^{i_h-f_h}\sum_{j=1}^{i_w-f_w}\frac{\partial L}{\partial O_{i,j}}\frac{\partial O_{i,j}}{\partial x_{m\prime,n\prime}} = \sum_{i=1}^{i_h-f_h}\sum_{j=1}^{i_w-f_w}\delta_{i,j}\frac{\partial O_{i,j}}{\partial x_{i+m',j+n\prime}} = \delta_{i,j}\otimes w_{m',n\prime}$$

مسئله ۳ –

قسمت آ) همانطور که در صورت سوال گفته شد، ماتریس ورودی را تبدیل به یک بردار ستونی کرده و آن را به این شکل در نظر میگیریم:

$$X = \begin{pmatrix} x_{0,0} \\ x_{0,1} \\ \vdots \\ x_{2,2} \end{pmatrix}$$

حال باید ماتریس w را تبدیل به یک ماتریس با ۴ سطر و ۹ ستون در نظر بگیریم چون نیاز به ۹ ستون برای حفظ شرط ضرب ماتریسی داریم و نیاز به ۴ سطر داریم که خروجی ضرب ماتریسی ، یک ماتریس *4*1* شود که سپس با ***reshape*** کردن آن به یک ماتریس *2*2* ، حاصل نهایی کانولوشن X و W را داشته باشیم. لذا ماتریس وزن را به شکل زیر تغییر میدهیم:

$$W = \begin{pmatrix} w_{0,0} & w_{0,1} & 0 & w_{1,0} & w_{1,1} & 0 & 0 & 0 & 0 \\ 0 & w_{0,0} & w_{0,1} & 0 & w_{1,0} & w_{1,1} & 0 & 0 & 0 \\ 0 & 0 & 0 & w_{0,0} & w_{0,1} & 0 & w_{1,0} & w_{1,1} & 0 \\ 0 & 0 & 0 & 0 & w_{0,0} & w_{0,1} & 0 & w_{1,0} & w_{1,1} \end{pmatrix}$$

حال ضرب ماتریسی معمولی زیر را محاسبه میکنیم:

$$Y = W \times X = \begin{pmatrix} w_{0,0}x_{0,0} + w_{0,1}x_{0,1} + w_{1,0}x_{1,0} + w_{1,1}x_{1,1} \\ w_{0,0}x_{0,1} + w_{0,1}x_{0,2} + w_{1,0}x_{1,1} + w_{1,1}x_{1,2} \\ w_{0,0}x_{1,0} + w_{0,1}x_{1,1} + w_{1,0}x_{2,0} + w_{1,1}x_{2,1} \\ w_{0,0}x_{1,1} + w_{0,1}x_{1,2} + w_{1,0}x_{2,1} + w_{1,1}x_{2,2} \end{pmatrix}$$

$$= \begin{pmatrix} w_{0,0}x_{0,0} + w_{0,1}x_{0,1} + w_{1,0}x_{1,0} + w_{1,1}x_{1,1} & w_{0,0}x_{0,1} + w_{0,1}x_{0,2} + w_{1,0}x_{1,1} + w_{1,1}x_{1,2} \\ w_{0,0}x_{1,0} + w_{0,1}x_{1,1} + w_{1,0}x_{2,0} + w_{1,1}x_{2,1} & w_{0,0}x_{1,1} + w_{0,1}x_{1,2} + w_{1,0}x_{2,1} + w_{1,1}x_{2,2} \end{pmatrix}$$

قسمت ب) ابتدا ماتریس X را به صورت یک وکتور ستونی در می آوریم یعنی:

$$X = \begin{pmatrix} 3 \\ 1 \\ 6 \\ 2 \end{pmatrix}$$

میدانیم که کانولوشن یک ورودی *4*4* با یک فیلتر *2*2* با *stride=1* برابر است با یک خروجی *2*2* . لذا ما باید ماتریس وزن مان را به شکل یک ماتریس *4*16* در بیاوریم و بعد *transpose* بگیریم که بشه یه ماتریس *16*4* و سپس ضرب ماتریسی این ماتریس را با بردار *X* حساب کنیم. اینجوری حاصل میشود یک بردار *16*1* و سپس با *reshape* کردن آن به ماتریس *4*4* که میخواستیم میرسیم.لذا:

$$W = \begin{pmatrix} 1 & 3 & 0 & 0 & 4 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 3 & 0 & 0 & 4 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 3 & 0 & 0 & 4 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 3 & 0 & 0 & 4 & 2 \end{pmatrix}$$

$$\rightarrow\ W^T = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 3 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 3 & 0 & 0 \\ 4 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 3 \\ 0 & 0 & 4 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 4 \\ 0 & 0 & 0 & 2 \end{pmatrix}$$

ادامه جواب صفحه بعد

حال ضرب ماتریسی ماتریس بالا در بردار X را حساب میکنیم:

$$i = W^T \times X = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 3 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 3 & 0 & 0 \\ 4 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 3 \\ 0 & 0 & 4 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 4 \\ 0 & 0 & 0 & 2 \end{pmatrix} \times \begin{pmatrix} 3 \\ 1 \\ 6 \\ 2 \end{pmatrix} = \begin{pmatrix} 3 \\ 9 \\ 1 \\ 3 \\ 12 \\ 6 \\ 4 \\ 2 \\ 6 \\ 18 \\ 2 \\ 6 \\ 24 \\ 12 \\ 8 \\ 4 \end{pmatrix} = \begin{pmatrix} 3 & 9 & 1 & 3 \\ 12 & 6 & 4 & 2 \\ 6 & 18 & 2 & 6 \\ 24 & 12 & 8 & 4 \end{pmatrix}$$

قسمت ج) ابتدا به تعریف یک *masked convolution* میپردازیم.ایده این کانولوشن این است که کرنل کانولوشن در برخی مکان ها صفر باشد که با اینکار بتوانیم فقط نقاط و مکان های خاصی از تصویر یا صوت را که سیگنال های مهم تری هستند را در پیش بینی ها تاثیر دهیم و با اینکار در اپلیکیشن های خاصی، دقت پیش بینی را بالا ببریم.همچنین در تصاویر رنگی میتوانیم این اولویت دهی را به غیر از بعد های مکانی، به کانال های رنگ نیز *extend* کنیم. یعنی یک سری کانال هارا بر کانال های دیگر اولویت دهیم و یا یک سری کانال هارا در مکان هایی نادیده بگیریم.با استفاده از *masked convolution* ها میتوانیم *color/channel splitting* نیز انجام دهیم. به شکلی که فیلتر های شبکه را به ۳ دسته تقسیم کنیم که هر دسته وظیفه اعمال کانولوشن روی کانال خاصی را داشته باشد.

محدودیت این نوع کانولوشن، در *dynamic* نبودن آن طبق تصاویر و اپلیکیشن های مختلف است. یعنی نمیتوان برای ورودی های مختلف، از ماسک های پویای مختلف استفاده کرد و نقاط مهم تصویر را در هر کیس متفاوت، متفاوت در نظر گرفت.محدودیت دیگر آن،*invariant* نبودن این ماسک ها به *Scaling* و *rotation* است که اگر تصویر ورودی هر کدام از این *transform* ها را داشته باشد، ماسک ما نیز باید تشخیص دهد که چون پویا نیست، از عهده این کار برنمی آید.

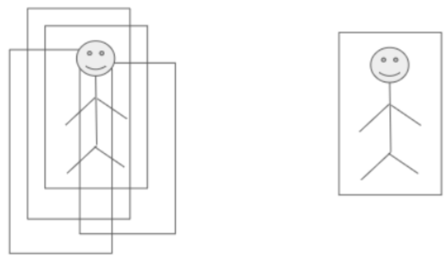
مسئله ۴ –

قسمت آ)در روش *YOLO* ، مسئله شناسایی اشیا را به شکل یک مسئله رگراسیون پیاده سازی کردند به شکلی که با یک بار پیمایش تصویر و ایجاد یک سری *bounding box* و محاسبه احتمال و *confidence* در آن باکس ها برا کلاس های مختلف اشیا، عمل شناسایی اشیا را انجام داده و به صورت *real time* عملیات بهینه سازی را نیز انجام میدهد. این روش آخر کار به صورت کاملا شهودی دسته بندی و شناسایی اشیا را گزارش میکند.در این روش کل اجزای *pipeline* روش های قبلی به عنوان یک عدد شبکه عصبی پیاده سازی شده است.در این روش تصویر به شبکه(*grid*) های *s*s* تقسیم بندی میشود و سپس اگر مرکز یک شئ در یکی از این *grid* ها قرار گرفت، آن *grid* مسئولیت شناسایی آن شی را به عهده دارد.هر *grid cell* مسئولیت ساخت *B* تا *bounding box* و محاسبه *confidence score* های آن باکس هارا بر عهده دارد.این امتیاز ها نشان میدهد که چقدر محتمل است که آن باکس شامل شئ مربوطه باشد و دقت لازم را داشته باشد.در لایه های اولیه کانولوشنی شبکه، ویژگی های تصویر ورودی استخراج شده و در لایه های کاملا متصل(*fully connected*) آخر، عملیات محاسبه و پیش بینی احتمالات صورت میگیرد.

مقایسه روش *YOLO* با روش *RCNN* :

۱-روش *RCNN* مبتنی بر متد پیشنهاد منطقه(*region proposal method*) است که ابتدا یک سری *bounding box* مهم شناسایی میکند و سپس از *classifier* ها برای این باکس ها استفاده میکند و سپس عملیات *preprocessing* را روی *bounding box* ها اعمال کرده که شناسایی های تکراری حذف شوند، باکس ها *tune* شوند و ... یعنی سیستم به صورت یک پایپ لاین پیچیده و با سرعت کم پیاده سازی شده است.بهینه سازی این روش سخت است چون سیستم شامل تعداد زیادی *component* است و هرکدام جداگانه باید بهینه شوند.در عوض روش *YOLO* به شکل یک مسئله رگراسیون است و کل سیستم یک پارچه عمل میکند و بخاطر اینکه فقط با یک بار پیمایش کلی تصویر ورودی، کار خود را انجام میدهد هم سریع تر است، هم بهینه سازی آن بخاطر یکپارچگی کل سیستم آسان تر است و هم در دقت عمل، بهتر از روش *RCNN* است.دلیل سرعت بالای *YOLO* هم این است که بر خلاف *RCNN* که مبتنی بر *pipeline* است، *YOLO* یک سیستم رگراسیون یکپارچه است.روش *RCNN* همانجور که گفته شد مبتنی بر *region proposal* است یعنی کل تصویر را یک باره و همزمان نگاه نمیکند اما *YOLO* کل تصویر را یکجا و موازی نگاه کرده و شناسایی را انجام میدهد.روش *RCNN* تعداد خیلی بیشتری *bounding box* ایجاد میکند که سیستم را کند و سنگین میکند. اما روش *YOLO* بخاطر اعمال محدودیت های مکانی روی *grid cell* ها ، تعداد کمتری *bounding box* ایجاد میکند.روش *RCNN* سرعت بالایی برای مسائل *real time* ندارد(بدلیل *pipeline* بودن و یکپارچه نبودن و سرعت پایین در ایجاد *bounding box* ها و شناسایی اشیا) اما روش *YOLO* سرعت بالایی دارد و میتوان در کاربرد های *real time* نیز از آن بهره برد.به عنوان آخرین مقایسه هم روش *RCNN* از معیار خطای محدودی در موقع *test* استفاده میکند ولی روش *YOLO* از معیار های بیشتری استفاده کرده و دقت خود را بالا میبرد.

قسمت ب) در روش *RCNN* و همچنین *YOLO* ، مشکلی که پیش می آید این است که چندین *bounding box* بتوانند یک شی را شناسایی کنند یعنی عملا مسئله *multiple detection* پیش می آید. حال ما هم بخاطر دقت هم بخاطر سرعت، باید بهترین *bounding box* را از میان همه انتخاب کنیم و شی مربوطه را فقط با آن *bounding box* شناسایی کنیم.در این هنگام، روش *non-maximal suppression* یا *NMS* به کمک ما می آید.در این روش از ایده *clustering* و معیار های محاسبه فاصله در این گونه مسائل مانند *k-means* و *nearest neighbour* و … استفاده میشود تا بتوانیم بهترین *box* را تشخیص دهیم.



Before NMS and after NMS

مشکلات و محدودیت های این روش: وقتی اشیا کوچک در تصویر به شکل گروهی پدید می آیند(مثلا دسته ای از کبوتران در حال پرواز) ، طبیعتا تعدادی *bounding box* آن هارا شناسایی میکنند و بخاطر اینکه چندین شی در اصل وجود دارد(هر پرنده یک شی محسوب میشود)، نمیتوان برای هر پرنده یک *box* پیدا کرد زیرا تعداد *box* ها در هر *grid cell* محدود است و بخاطر همین اشیا کوچک گروهی به عنوان یک شی شناسایی میشوند و این اشتباه است.

مشکل دیگر این الگوریتم، توانایی انتخاب *threshold* مناسب است زیرا انتخاب این آستانه روی دقت مدل تاثیر زیادی دارد.در الگوریتم معمولی *NMS* اگر یک *box* مقدار *IOU* بیشتر از آستانه داشته باشد اما *confidence* بالایی داشته باشد، حذف میشود و ممکن است یک باکس با *IOU* کمتر از حد آستانه ولی *confidence* پایین که بهتر بود حذف شود، نگه داشته شود. الگوریتم *nms* معمولی به شکل زیر است:

Algorithm 1 Non-Max Suppression

```
procedure NMS(B,c)
    B_nms ← ∅    Initialize empty set
    for b_i ∈ B do  =>  Iterate over all the boxes
        discard ← False    Take boolean variable and set it as false. This variable indicates whether b(i) should be kept or discarded
        for b_j ∈ B do    Start another loop to compare with b(i)
            if same(b_i, b_j) > λ_nms then    If both boxes having same IOU
                if score(c, b_j) > score(c, b_i) then
                    discard ← True    Compare the scores. If score of b(i) is less than that of b(j), b(i) should be discarded, so set the flag to True.
        if not discard then
            B_nms ← B_nms ∪ b_i    Once b(i) is compared with all other boxes and still the discarded flag is False, then b(i) should be considered. So add it to the final list.
    return B_nms    Do the same procedure for remaining boxes and return the final list
```

مشکل دوم: کارایی و سرعت روش *nms* در کاربرد های *real time* ضعیف است چون حریصانه عمل میکند و پیچیدگی اش *n**2* است.

قسمت ج) برای رفع مشکل دوم گفته شده، از ایده ای به آدرس https://github.com/bharatsingh430/soft-nms استفاده میکنیم که روش *soft nms* است. در این روش، به جای حذف کامل *box* های با *IOU* بالای حد آستانه، آن را نگه میداریم اما به نسبت *IOU* آن، از *confidence* اش کم میکنیم. الگوریتم *soft nms* به شکل زیر است:

Input : $\mathcal{B} = \{b_1, .., b_N\}$, $\mathcal{S} = \{s_1, .., s_N\}$, N_t
$\mathcal{B}$ is the list of initial detection boxes
$\mathcal{S}$ contains corresponding detection scores
N_t is the NMS threshold

begin
$\quad \mathcal{D} \leftarrow \{\}$
$\quad$ **while** $\mathcal{B} \neq$ *empty* **do**
$\quad\quad m \leftarrow \text{argmax}\ \mathcal{S}$
$\quad\quad \mathcal{M} \leftarrow b_m$
$\quad\quad \mathcal{D} \leftarrow \mathcal{D} \bigcup \mathcal{M}; \mathcal{B} \leftarrow \mathcal{B} - \mathcal{M}$
$\quad\quad$ **for** b_i *in* $\mathcal{B}$ **do**
$\quad\quad\quad$ **if** $iou(\mathcal{M}, b_i) \geq N_t$ **then**
$\quad\quad\quad\quad \mathcal{B} \leftarrow \mathcal{B} - b_i; \mathcal{S} \leftarrow \mathcal{S} - s_i$
$\quad\quad\quad$ **end**
$\quad\quad\quad$ NMS
$\quad\quad\quad s_i \leftarrow s_i f(iou(\mathcal{M}, b_i))$
$\quad\quad\quad$ Soft-NMS
$\quad\quad$ **end**
$\quad$ **end**
$\quad$ **return** $\mathcal{D}, \mathcal{S}$
end

برای حل مشکل ۲ ، در
http://openaccess.thecvf.com/content_CVPR_2019/papers/Cai_MaxpoolNMS_Getting_Rid_of_NMS_Bottlenecks_in_Two-Stage_Object_Detectors_CVPR_2019_paper.pdf

از روش *maxpoolNMS* استفاده میشود که در دو *Stage* ، با موازی سازی و استفاده از رویکرد *maxpooling* ، سرعت عمل *nms* های قبلی رو افزایش داده(حدود ۲۰ برابر سریع تر) نسبت به *nms* معمولی(*greedy*)

مسئله ۵ –

قسمت آ) از رویکرد پایین به بالا جلو میرویم یعنی:

$$x_{l+1} = F(x_l) + x_l \qquad , \qquad x_{l+2} = F(x_{l+1}) + x_{l+1}$$

حال دو طرف رابطه بالا را باهم جمع میکنیم:

$$x_{l+1} + x_{l+2} = F(x_l) + x_l + F(x_{l+1}) + x_{l+1}$$

حال با استقرا به نتیجه زیر میرسیم:

$$x_{l+1}+x_{l+2}+\cdots+x_{L-1}+x_L=F(x_l)+x_l+F(x_{l+1})+x_{l+1}+\cdots+\ F(x_{L-1})+x_{L-1}$$

لذا داریم:

$$X_L=x_l+\left(F(x_l)+F(x_{l+1})+\cdots+\ F(x_{L-1})\right)=x_l+\sum\nolimits_{i=l}^{L-1}F(x_i)$$

قسمت ب) طبق قاعده زنجیره ای میدانیم:

$$\frac{\partial E}{\partial x_l}=\ \frac{\partial E}{\partial x_L}\frac{\partial x_L}{\partial x_l}=\ \frac{\partial E}{\partial x_L}\left(x_l+\ \sum\nolimits_{i=l}^{L-1}\frac{\partial}{\partial x_l}F(x_i)\right)$$

حال میدانیم که :

$$\frac{\partial}{\partial x_l}F(x_i)=\ \frac{\partial F(x_i)}{\partial x_i}\frac{\partial x_i}{\partial x_{i-1}}\ldots\ \frac{\partial x_{l+1}}{\partial x_l}=F'(x_i)(F'(x_{i-1}))\ldots(F'(x_l)+1)$$
$$=\ F'(x_i)\prod_{j=l}^{i-1}\left[F'\left(x_j\right)+1\right]$$

با تجمیع دو رابطه بدست آمده بالا داریم:

$$\frac{\partial E}{\partial x_l}=\ \frac{\partial E}{\partial x_L}\left(1+F'(x_l)+\sum\nolimits_{i=l}^{L-1}\left[F'(x_i)\prod_{j=l}^{i-1}\left[F'\left(x_j\right)+1\right]\right]\right)$$

قسمت ج) در *backprob* معمولی ما ضرب یک سری مشتق توی هم داشتیم و همین باعث این میشد که وقتی عمق شبکه را زیاد میکنیم، این ضرب ها به سمت صفر میل کنند و به عبارتی محوشدن گرادیان رخ دهد. اما در این شبکه، $\frac{\partial E}{\partial x_l}$ یک چیز در خود دارد که از محو شدن گرادیان جلوگیری میکند و آن مجموع *F’* هاست.لذا هرچه شبکه را عمیق کنیم، گرادیان هیچ وقت به سمت صفر میل نمیکند.

قسمت د) گزارش های خواسته شده در نوت بوک آورده شده است.

قسمت ه) نمودار تابع و دقت های خواسته شده در نوت بوک آورده شده است.

مسئله ۶ – در نوت بوک گزارش ها آورده شده است.