تمرین سری دوم

یادگیری ژرف

امیرحسین محمدی

۹۹۲۰۱۰۸۱

مسئله‌ی ۱. Network Design

(آ) فرض کنید که شبکه ای با ساختار زیر داریم، خروجی هر لایه و تعداد پارامترهای آن را محاسبه کنید.

$$Input = ۲۲۴ * ۲۲۴ * ۳ \rightarrow ۱$$
$$۱ \rightarrow Conv[۳ * ۳ * ۶۴, stride = ۱, pad = ۱] \rightarrow ۲$$
$$۲ \rightarrow ReLU \rightarrow ۳$$
$$۳ \rightarrow Pooling[۲ * ۲ * ۳۲, stride = ۲] \rightarrow ۴$$
$$۴ \rightarrow FC[۱۰ - class] \rightarrow ۵$$

اعداد ۱ تا ۴ برای مشخص کردن خروجی و ورودی های لایه ها استفاده شده اند.

(ب) درصورتیکه بخواهیم فقط با یک لایه ی تمام متصل (FC) تصویری با ابعاد ورودی و خروجی مشابه داشته باشیم، تعداد پارامترهای لایه ی تمام متصل را محاسبه کنید

(ج) تفاوت Deformable Convolution با Standard Convolution در چیست و چه مزایایی نسبت به حالت استاندارد دارد؟

$$Input = ۲۲۴ * ۲۲۴ * ۳ \rightarrow ۱$$

$$۱ \rightarrow Conv[۳ * ۳ * ۶۴, stride = ۱, pad = ۱] \rightarrow ۲$$

ابعاد خروجی لایه ی کانوولوشنی داریم:

$$output = \frac{input - kernel_{size} + 2 * padding}{stride} + 1$$

$$output = \frac{224 - 3 + 2 * 1}{1} + 1 = 224$$

$$output = (224,224,64)$$

تعداد پارامترهای تولید شده بعد از لایه ی کانوولوشنی داریم:

parameters=((shape of width of filter*shape of height filter*number of filters in the previous layer+1)*number of filters)

$$Parameters = (3 * 3 * 3 + 1) * 64 = 1792$$

$$۲ \rightarrow ReLU \rightarrow ۳$$

تعداد پارامترهای تولید شده

ابعاد خروجی

$Parameters = o$

$output = (224,224,64)$

توابع فعالیت صرفا سطح بالاتری از ویژگی ها را تولید می کنند و پارامتر اضافه تری تولید نمی کنند

ابعاد تغییری نمی کنند و زیرا صرفا به ازای هر ورودی مجددا یک خروجی تولید می شود

$$۳ \rightarrow Pooling[۲ * ۲ * ۳۲, stride = ۲] \rightarrow ۴$$

ابعاد خروجی بعد از لایه ی pooling

تعداد پارامترهای تولید شده بعد از لایه ی pooling

$$output = \frac{input - kernel_{size}}{stride} + 1$$

$$Parameters = o$$

$$output = \frac{224 - 2}{2} + 1 = 112$$

$$output = (112,112,64)$$

لایه pooling صرفا یک تابع است که ویژگی هایی را استخراج می کند و بنابراین پارامتری تولید نمی کند

$$۴ \rightarrow FC[۱۰ - class] \rightarrow ۵$$

تعداد پارامترهای تولید شده با ازای لایه های کاملا متصل

ابعاد خروجی

$output = (10)$

((current layer neurons c * previous layer neurons p)+1*c)

$Parameters = (112 * 112 * 64) * 10 + 10 = 8028170$

با توجه به فرضیات مسئله

(ب) درصورتیکه بخواهیم فقط با یک لایه ی تمام متصل (FC) تصویری با ابعاد ورودی و خروجی مشابه داشته باشیم، تعداد پارامترهای لایه ی تمام متصل را محاسبه کنید

$$Input = ۲۲۴ * ۲۲۴ * ۳ \rightarrow ۱$$

$$1 \rightarrow FC[۱۰ - class] \rightarrow 2$$

تعداد پارامترهای تولید شده

ابعاد خروجی

$output = (10)$

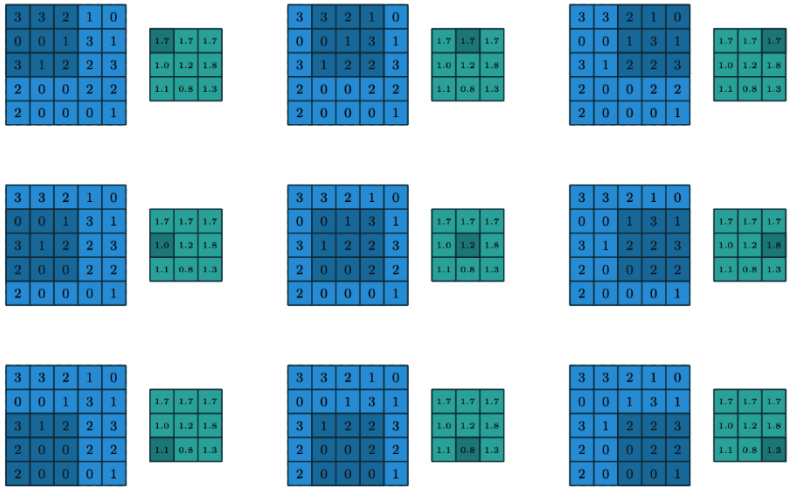
((current layer neurons c * previous layer neurons p)+1*c)

$$Parameters = (224 * 224 * 3) * 10 + 10 = 1505290$$

(ج) *تفاوت* Deformable Convolution *با* Standard Convolution *در چیست و چه مزایایی نسبت به حالت استاندارد دارد؟*

Standard Convolution

در کانوولوشن استاندارد ما یک کرنل با ابعاد ثابت داریم که با گام های ثابت حرکت می کند و ویژگی های تصاویر را استخراج می کند.
در واقع نقاط کانوولوشنی ما به صورت منظم قرار دارند. که رابطه ی آن به صورت زیر تعریف می شود.

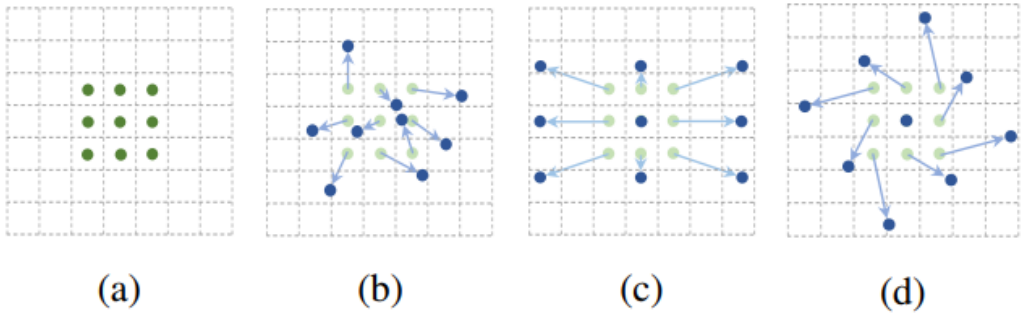


$$\mathbf{y}(\mathbf{p}_0) = \sum_{\mathbf{p}_n \in \mathcal{R}} \mathbf{w}(\mathbf{p}_n) \cdot \mathbf{x}(\mathbf{p}_0 + \mathbf{p}_n)$$

این مدل استخراج ویژگی گاهی به دلیل وِیژگی ها و اسکیل ها متفاوت object ها در تصاویر به خوبی عمل نمی کند. زیرا همانطور که گفته شد ابعاد و تعداد گام ها در این مدل یکسان است و از طرفی به دلیل اینکه در این مدل از کانوولوشن کرنل های ما شکل مربعی دارند لزوما object های با اشکال گوناگون به خوبی شناسایی نمی شوند (زیرا ما تصاویر و object های با اسکیل های متفاوت داریم) . بنابراین ما نیاز داریم تا کانوولوشن هایی اعمال کنیم که شکل های با عرض و ارتفاع های مختلف و .... را شناسایی کنیم.

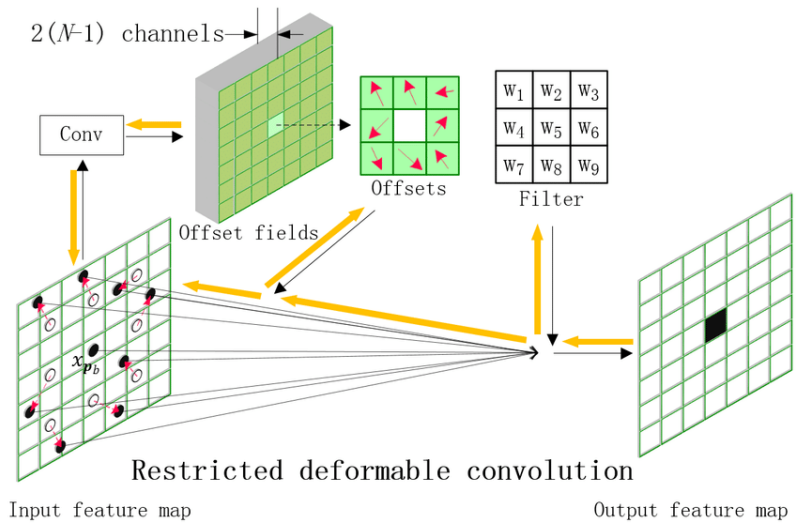
Deformable Convolution

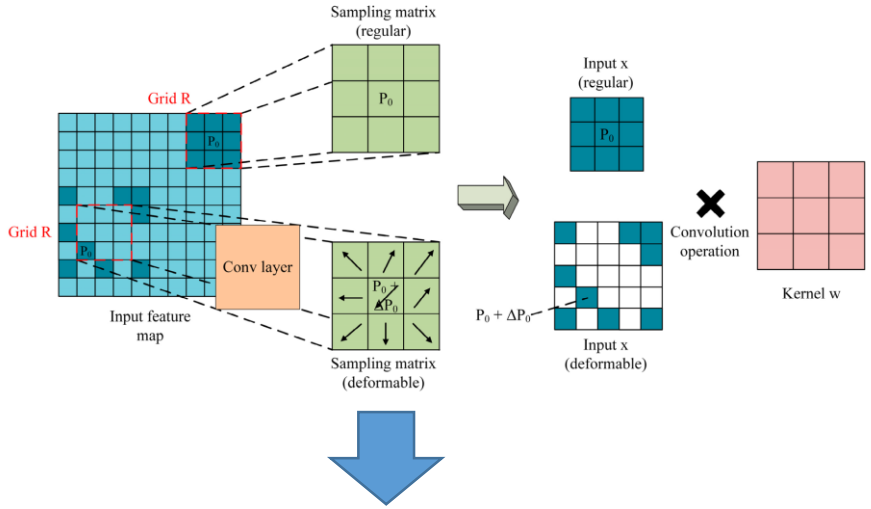
با توجه به مشکلات مطرح شده در کانوولوشن های استاندارد، ایده استفاده از کانوولوشن های شکل پذیر ایجاد شد به گونه ای که بتوانند برخلاف شکل مربعی و ثابتی که در فرم استاندارد وجود داشت Object های با شکل های پیچده تر را نیز شناسایی کنند. برای این کار کافی بود تا به معادله ی محاسبه ی کانوولشن که در کانوولوشن استاندارد به آن پرداخته شد یک پارامتر قابل یادگیری اضافه کنیم که در هنگام آموزش شبکه به روزرسانی می شود و شکل کانوولوشن را تغییر داده و به نوعی کانوولوشن شکل پذیر ایجاد می کند که رابطه ی آن به شکل زیر تعریف می شود.



$$\mathbf{y}(\mathbf{p}_0) = \sum_{\mathbf{p}_n \in \mathcal{R}} \mathbf{w}(\mathbf{p}_n) \cdot \mathbf{x}(\mathbf{p}_0 + \mathbf{p}_n + \Delta\mathbf{p}_n).$$

همانطور که می بینیم بر اساس پارامتر ΔP_n که به معادله ی کانوولوشن اضافه کردیم کانوولوشن های ما همانند شکل های بالا شکل پذیر شده اند مثلا در ابتدای آموزش شکل کانوولوشن همانند کانوولشن استاندارد است (همانطور که در شکل a می بینیم) اما در هنگام یادگیری شکل کانوولشن تغییر می کند و مثلا در شکل c همانطور که می بینیم شکل کانوولوشن در راستای محور ایکس رشد می کند و مثلا می توانند اشکال عریض مثل اتوبوس را شناسایی کند. تفاوت کانوولوشن استاندارد و کانوولوشن شکل پذیر در شکل مقابل مشخ است.





در شکل مقابل مثال هایی از کانوولوشن های شکل پذیر را می بینیم که ابعاد و شکل کانوولوشن متناسب با Object های موجود در تصویر تغییر کرده است. نتایج نشا می دهد که استفاده از کانوولوشن های شکل پذیر در مسائلی که شناسایی object ها اهمیت دارند بسیار مفید و موثر خواهند بود. همچنین استفاده از کانوولوشن های شکل پذیر تعداد پارامتر ها ویژگی های زیادی را به شبکه تحمیل نمی کند. همچنین در مسائل Object detection بسیار بهتر عمل میکند. همچنین در مسائلی که تصاویر اسکیل های زیادی دارند بهتر عمل می کنند. نمونه ای از برترها و مقایسه های بین کانوولوشن استاندارد و کانوولوشن شکل پذیر (deformable) بر اساس بررسی شبکه های مختلف در زیر آورده شده است.

Test Accuracy	Regular CNN	Deformable CNN
Regular MNIST	98.74%	97.27%
Scaled MNIST	57.01%	92.55%

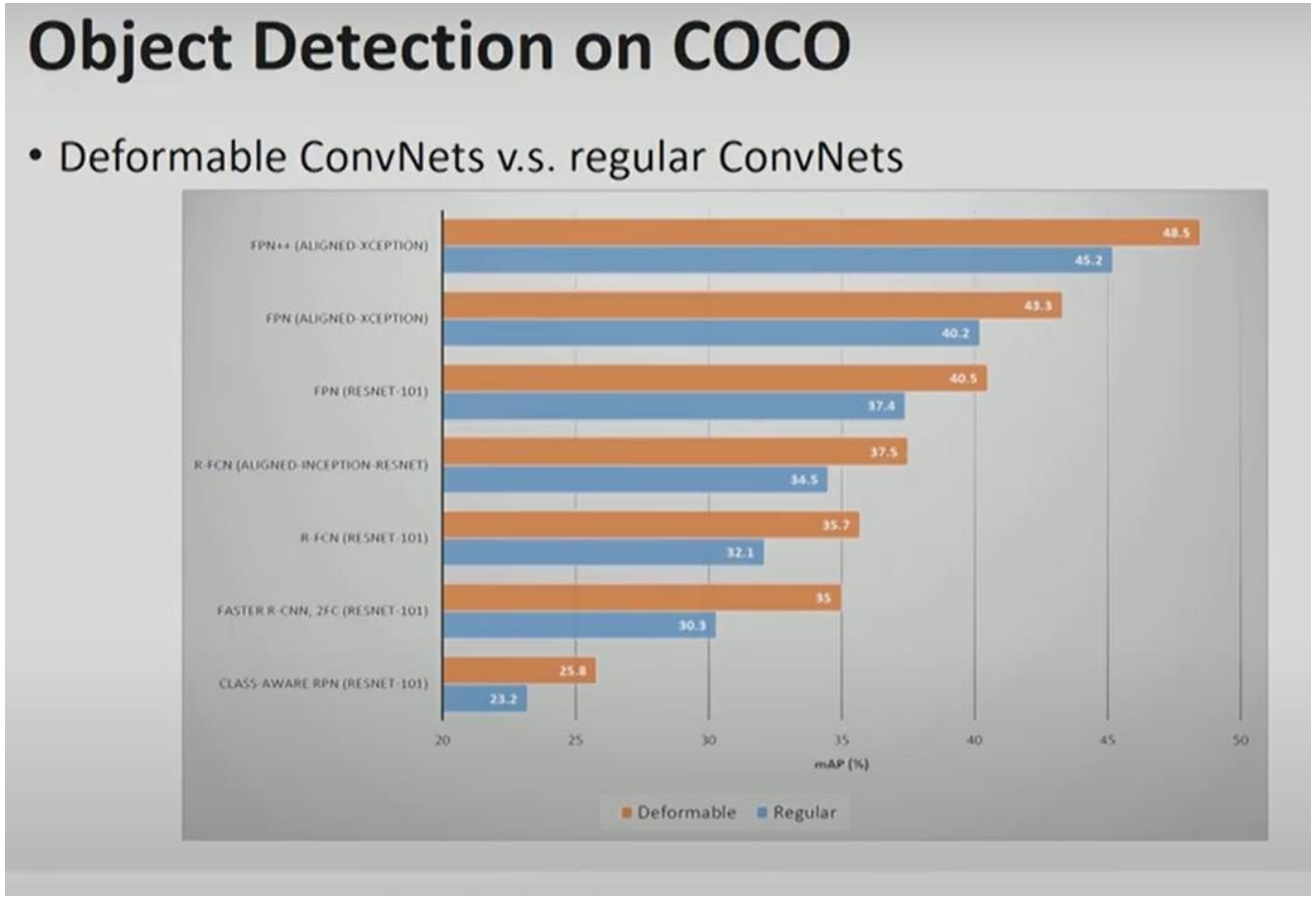
مقایسه ای از عملکرد کانولوشن های استاندارد و کانوولوشن های شکل پذیر بر روی مجموعه دادگان MNIST که اسکیل شده اند و نشده اند.

deformation modules	DeepLab mIoU@V / @C	class-aware RPN mAP@0.5 / @0.7	Faster R-CNN mAP@0.5 / @0.7	R-FCN mAP@0.5 / @0.7
atrous convolution (2,2,2) (default)	69.7 / 70.4	68.0 / 44.9	78.1 / 62.1	80.0 / 61.8
atrous convolution (4,4,4)	73.1 / 71.9	72.8 / 53.1	78.6 / 63.1	80.5 / 63.0
atrous convolution (6,6,6)	73.6 / 72.7	73.6 / 55.2	78.5 / 62.3	80.2 / 63.5
atrous convolution (8,8,8)	73.2 / 72.4	73.2 / 55.1	77.8 / 61.8	80.3 / 63.2
deformable convolution	**75.3 / 75.2**	**74.5 / 57.2**	78.6 / 63.3	81.4 / 64.7
deformable RoI pooling	N.A	N.A	78.3 / 66.6	81.2 / 65.0
deformable convolution & RoI pooling	N.A	N.A	**79.3 / 66.9**	**82.6 / 68.5**

ارزیابی کانوولوشن های شکل پذیر با روش های دیگر مثل کانوولوشن های atrous بر روی شبکه ی ResNet.
همانطور که مشخص است کانوولوشن های Deformable به دقت بهتری رسیده اند.

method	# params	net. forward (sec)	runtime (sec)
DeepLab@C	46.0 M	0.610	0.650
Ours	46.1 M	0.656	0.696
DeepLab@V	46.0 M	0.084	0.094
Ours	46.1 M	0.088	0.098
class-aware RPN	46.0 M	0.142	0.323
Ours	46.1 M	0.152	0.334
Faster R-CNN	58.3 M	0.147	0.190
Ours	59.9 M	0.192	0.234
R-FCN	47.1 M	0.143	0.170
Ours	49.5 M	0.169	0.193

همانطور که مشخص است کانوولوشن های Deformable بر اساس تعداد پارمترها و زمان اجرا و پیچیدگی محاسباتی سربار محسباتی و حافظه ای چشم گیری به شبکه تحمیل نمی کند.



همانطور که مشخص است در مسائل تشخیص شی هم، کانوولشن های شکل پذیر نتایج بهتری نسبت به کانوولوشن های استاندارد دارند.

مسئله‌ی ۲. Transposed Convolution

کانولوشن بین ورودی X و فیلتر W به صورت زیر است:

$$X = \begin{bmatrix} x_{(۰,۰)} & x_{(۰,۱)} & x_{(۰,۲)} \\ x_{(۱,۰)} & x_{(۱,۱)} & x_{(۱,۲)} \\ x_{(۲,۰)} & x_{(۲,۱)} & x_{(۲,۲)} \end{bmatrix} \quad W = \begin{bmatrix} w_{(۰,۰)} & w_{(۰,۱)} \\ w_{(۱,۰)} & w_{(۱,۱)} \end{bmatrix}$$

می توان عملیات کانولوشن را به صورت ضرب ماتریسی نوشت که ورودی و خروجی را به صورت یک بردار و فیلتر را به صورت یک ماتریس در نظر گرفت. ورودی X را به صورت بردار زیر نمایش می دهیم:

$$X = \begin{bmatrix} x_{(۰,۰)} & x_{(۰,۱)} & \dots & x_{(۲,۲)} \end{bmatrix}^T$$

(آ) عملیات کانولوشن بالا با $S = ۱$ را به صورت ضرب ماتریسی $Y = AX$ بنویسید.

(ب) با استفاده از نمایش ماتریسی بالا می توان گرادیان پس انتشار نسبت به ورودی X، را به صورت $\frac{\partial L}{\partial X} = A^T \frac{\partial L}{\partial Y}$ نمایش داد.
عملیات transposed convolution را می توان مشابه عملیات گرادیان پس انتشار نسبت به ورودی درنظر گرفت و می توان این عملیات را به صورت کانولوشن مستقیم با ماتریس A^T به عنوان فیلتر دانست.
فرض کنید X خروجی یک عملیات کانولوشن با فیلتر W و $\text{stride} = 2$ است.
خروجی transposed convolution را با ورودی های زیر محاسبه کنید.

$$X = \begin{bmatrix} ۲ & ۱ \\ ۳ & ۶ \end{bmatrix} \quad W = \begin{bmatrix} ۱ & ۲ \\ ۴ & ۱ \end{bmatrix}$$

❖ طبق فرضیات مسئله داریم:

$$X = \begin{bmatrix} x_{(0,0)} & x_{(0,1)} & x_{(0,2)} \\ x_{(1,0)} & x_{(1,1)} & x_{(1,2)} \\ x_{(2,0)} & x_{(2,1)} & x_{(2,2)} \end{bmatrix} \quad W = \begin{bmatrix} w_{(0,0)} & w_{(0,1)} \\ w_{(1,0)} & w_{(1,1)} \end{bmatrix}$$

❖ می توانیم ماتریس ورودی را به صورت زیر تعریف کنیم:

$$X = [x_{0,0}, \quad x_{0,1}, \quad x_{0,2}, \quad x_{1,0}, \quad x_{1,1}, \quad x_{1,2}, \quad x_{2,0} \quad x_{2,1} \quad x_{2,2}]^T{}_{1*9}$$

همانطور که می ببینم ابعاد ماتریس X به صورت ۱ در ۹ است. برای اینکه ابعاد ماتریس w یا همان کرنل را بدست آوریم، می دانیم در تبدیل عملیات کانوولوشن به ضرب ماتریسی اگر ماتریس ورودی (x) n در n باشد و ابعاد ماتریس کرنل (w) m در m باشد آنگاه برای اینکه بتوانیم ماتریس x مسئله ما که ابعاد ۱ در ۹ دارد بتواند با ما ماتریس w ضرب شود ابعاد ماتریس A که در خود به نوعیی اطلاعات ماتریس w را دارد به ابعاد n^2 در $(n-m+1)^2$ به تغییر کند یعنی ما باید یک ماترس به نام A با ابعاد ۴ در ۹ داریم. بنابراین باید این چهار مقدار ماتریس w به نوعی در ماتریس جدید که با ابعاد ۴ در ۹ وجود دارد قرار گیرد که بتواند دقیقا عملیات کانوولوشن را شبیه سازی کند اما به صورت ضرب ماتریسی، همچین با توجه به stride=1 باید مقادیر وزن ها طوری در ماتریس A قرار گیرند که این شرط نیز لحاظ شود، بنابراین ماتریس A با ابعاد ۴ در ۹ به صورت زیر تعریف می شود:

$$A = \begin{bmatrix} w_{0,0} & w_{0,1} & 0 & w_{1,0} & w_{1,1} & 0 & 0 & 0 & 0 \\ 0 & w_{0,0} & w_{0,1} & 0 & w_{1,0} & w_{1,1} & 0 & 0 & 0 \\ 0 & 0 & 0 & w_{0,0} & w_{0,1} & 0 & w_{1,0} & w_{1,1} & 0 \\ 0 & 0 & 0 & 0 & w_{0,0} & w_{0,1} & 0 & w_{1,0} & w_{1,1} \end{bmatrix}_{4*9}$$

ماتریس بالا بر اساس توضیح مسئله مثل stride=1 و ابعاد ماتریس کانوولوشن ایجاد شده است.

خواسته ی مسئله

❖ $y = AX$

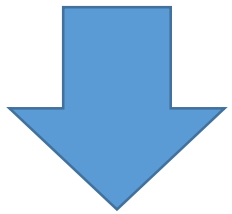
$$A = \begin{bmatrix} w_{0,0} & w_{0,1} & 0 & w_{1,0} & w_{1,1} & 0 & 0 & 0 & 0 \\ 0 & w_{0,0} & w_{0,1} & 0 & w_{1,0} & w_{1,1} & 0 & 0 & 0 \\ 0 & 0 & 0 & w_{0,0} & w_{0,1} & 0 & w_{1,0} & w_{1,1} & 0 \\ 0 & 0 & 0 & 0 & w_{0,0} & w_{0,1} & 0 & w_{1,0} & w_{1,1} \end{bmatrix}_{4*9}$$

$$X = [x_{0,0}, \ x_{0,1}, \ x_{0,2}, \ x_{1,0}, \ x_{1,1}, \ x_{1,2}, \ x_{2,0} \ x_{2,1} \ x_{2,2}]^T_{1*9}$$

$$y = \begin{bmatrix} w_{0,0} & w_{0,1} & 0 & w_{1,0} & w_{1,1} & 0 & 0 & 0 & 0 \\ 0 & w_{0,0} & w_{0,1} & 0 & w_{1,0} & w_{1,1} & 0 & 0 & 0 \\ 0 & 0 & 0 & w_{0,0} & w_{0,1} & 0 & w_{1,0} & w_{1,1} & 0 \\ 0 & 0 & 0 & 0 & w_{0,0} & w_{0,1} & 0 & w_{1,0} & w_{1,1} \end{bmatrix}_{4*9} * \begin{bmatrix} x_{0,0} \\ x_{0,1} \\ x_{0,2} \\ x_{1,0} \\ x_{1,1} \\ x_{1,2} \\ x_{2,0} \\ x_{2,1} \\ x_{2,2} \end{bmatrix}_{9*1} = \begin{bmatrix} w_{0,0} * x_{0,0} + w_{0,1} * x_{0,1} + w_{1,0} * x_{1,0} + w_{1,1} * x_{1,1} \\ w_{0,0} * x_{0,1} + w_{0,1} * x_{0,2} + w_{1,0} * x_{1,1} + w_{1,1} * x_{1,2} \\ w_{0,0} * x_{1,0} + w_{0,1} * x_{1,1} + w_{1,0} * x_{2,0} + w_{1,1} * x_{2,1} \\ w_{0,0} * x_{1,1} + w_{0,1} * x_{1,2} + w_{1,0} * x_{2,1} + w_{1,1} * x_{2,2} \end{bmatrix}_{4*1}$$

❖ خروجی ما در حال حاضر متناسب با ابعاد خروجی کانوولوشن نیست (به خاطر تغییر سایزهایی که دادیم) بنابراین با خروجی ما ریشپ شود (۲ در ۲) تا دقیقا خروجی حاصل ضرب بالا با خروجی کانوولوشن برابر شود.

بنابراین پس از reshape
خواهیم داشت:



$$y = AX = \begin{bmatrix} w_{0,0} * x_{0,0} + w_{0,1} * x_{0,1} + w_{1,0} * x_{1,0} + w_{1,1} * x_{1,1} & w_{0,0} * x_{0,1} + w_{0,1} * x_{0,2} + w_{1,0} * x_{1,1} + w_{1,1} * x_{1,2} \\ w_{0,0} * x_{1,0} + w_{0,1} * x_{1,1} + w_{1,0} * x_{2,0} + w_{1,1} * x_{2,1} & w_{0,0} * x_{1,1} + w_{0,1} * x_{1,2} + w_{1,0} * x_{2,1} + w_{1,1} * x_{2,2} \end{bmatrix}_{2*2}$$

(ب) با استفاده از نمایش ماتریسی بالا می توان گرادیان پس انتشار نسبت به ورودی X، را به صورت $\frac{\partial L}{\partial X} = A^T \frac{\partial L}{\partial Y}$ نمایش داد.

عملیات transposed convolution را می توان مشابه عملیات گرادیان پس انتشار نسبت به ورودی درنظر گرفت و می توان این عملیات را به صورت کانولوشن مستقیم با ماتریس A^T به عنوان فیلتر دانست.

فرض کنید X خروجی یک عملیات کانولوشن با فیلتر W و stride = 2 است.
خروجی transposed convolution را با ورودی های زیر محاسبه کنید.

$$X = \begin{bmatrix} ۲ & ۱ \\ ۳ & ۶ \end{bmatrix} \quad W = \begin{bmatrix} ۱ & ۲ \\ ۴ & ۱ \end{bmatrix}$$

❖ با توجه به فرضیات مسئله و نتایج قسمت الف داریم:

$$x = [2 \quad 1 \quad 3 \quad 6]_{1*4}$$

$$w = \begin{bmatrix} 1 & 2 & 0 & 0 & 4 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 2 & 0 & 0 & 4 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 0 & 0 & 4 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 0 & 0 & 4 & 1 \end{bmatrix}_{4*16}$$

با توجه به stride =2 و ابعاد ماتریس های x و w

❖ به دلیل اینکه طبق فرضیات x خروجی مسئله است بنابراین باید w را بر اساس سایز ورودی مسئله طراحی کنیم برای بدست آوردن سایز ورودی مسئله داریم:

$$\frac{input - 2 * kernel_{size}}{stride} + 1 = 2$$

بنابراین ابعاد و.رودی مسئله ۴در ۴ بوده است

بنابراین همانطور گه در قسمت قبل نیز اشاره شده برای بدست آوردن ابعاد w با ابعاد آنرا به گونه ای در نظر بگیریم که مقدار خروجی آن ۱۶ باشد تا با reshape کردن بتوانیم ویژگی هایی با ابعاد سایز ورودی تولید کنیم. بنابراین تعاد ستون های ماتریس w ۱۶ و تعداد سطرهای آن ۴ در نظر گرفته می شود(تا بتوانیم ضرب ماتریسی انجام دهیم). همچینین به دلیل اینکه stride ۲ در نظر گرفته شده بین هر دو عدد غیر از صفر دو صفر فاصله می اندازیم(برای ایجاد stride)

$$[2 \quad 1 \quad 3 \quad 6]_{1*4} * \begin{bmatrix} 1 & 2 & 0 & 0 & 4 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 2 & 0 & 0 & 4 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 0 & 0 & 4 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 0 & 0 & 4 & 1 \end{bmatrix}_{4*16} = [2,4,1,2,8,2,4,1,3,6,6,12,12,3,24,6]_{1*16}$$

همانطور که گفتیم باید متناسب با ابعاد ورودی بردار حاصل را reshape کنیم

$$Transposed\ Convolution = \begin{bmatrix} 2 & 4 & 1 & 2 \\ 8 & 2 & 4 & 1 \\ 3 & 6 & 6 & 12 \\ 12 & 3 & 24 & 6 \end{bmatrix}_{4*4}$$

مسئله‌ی ۳. Object Detection

یکی از کاربرد های شبکه های کانولوشن، آشکارسازی اشیا است. مقالات مرتبط را مطالعه کرده و به سوالات زیر پاسخ دهید.

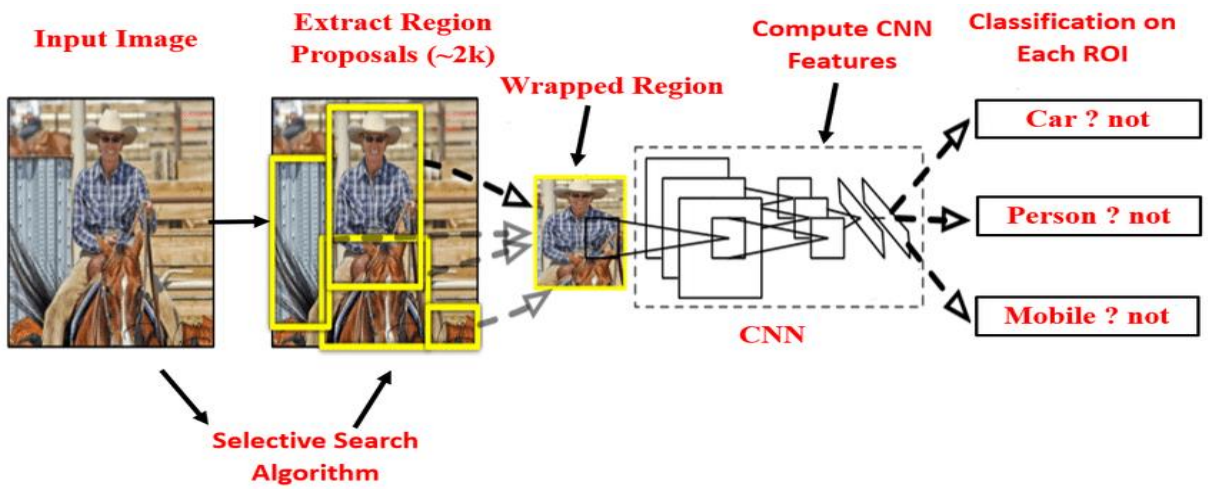
(آ) روش های YOLO و RCNN و RFCN را بررسی کرده و با یکدیگر مقایسه کنید.

(ب) در آشکارسازی های real-time استفاده از کدام یک را پیشنهاد می کنید.

(ج) لایه Spatial Pyramid Pooling را مختصرا توضیح داده و مزایای استفاده از آن را، در آشکارسازی تصاویر بیان کنید.

توضیحی در مورد RCNN

روش RCNN یک روش برای شناسایی Object های موجود در یک تصویر است (Object detection). بر اساس این روش یک تصویر به نواحی مختلف تقسیم شده (Region proposal) و از یک الگوریتم به اسم selective search برای ایجاد این نواحی استفاده می کند. این نواحی که تقریبا طبق گفته مقاله ۲۰۰۰ ناحیه متفاوت ROI (regions of interest)هستند به شبکه ی کانوولوشنی مورد نظر به عنوان ورودی داده می شود مثل Alex net از این شبکه با حجم قابل تجهی از داده های ورودی مواجه می شود برای همین معمولا دیتاست های معمول برای object detection حجم کمی دارند مثلا coco ۹۰۰۰ داده دارد. همانطور که در شکل زیر می بینیم، همچنین تابع هدف آن به صورت زیر تعریف می شود:



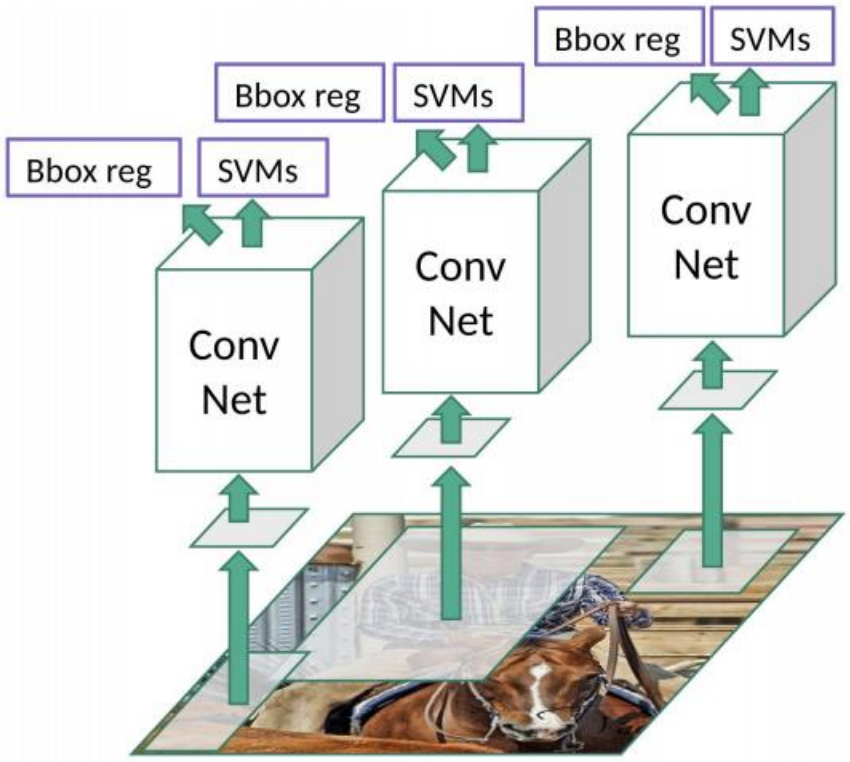
$$L(\{p_i\},\{t_i\}) = \frac{1}{N_{cls}}\sum_i N_{cls}(p_i, {p_i}^*) + \lambda\frac{1}{N_{reg}}\sum_i {P_i}^* L_{reg}(t_i, t_i*)$$

که در این فرمول P_i احتمال پیش بینی شده است، P_i^* احتمال واقعی است که در صورتی که object وجود داشته باشد مقدار ۱ و در صورتی که object وجود نداشته باشد مقدار ۰ به خود می گیرد. t_i مقدار مختصات جعبه های محاطی است و t_i^* مقدار واقعی جعبه های محاطی است. L_{cls} مقدار لاگ خطای دسته بندی است و L_{reg} خطای رگرسیون است.

توضیحی در مورد RCNN

این روش بر اساس یک سری جعبه های محاط شده bounding box استفاده می کند و بر اساس مدل رگرسیونی مکانی که به عنوان مختصات جعبه محاطی پیش بینی کرده را با جعبه محاطی واقعی مقایسه می کند و مقدار تابع هزینه را محاسبه می کند. برای تعیین میزان همپوشانی مقدار جبعبه محاطی تخیمن زده شده با جبعه محاطی واقعی از معیار intersection-over-union(IoU) استفاده می شود.
این الگوریتم از سه بخش اصلی تشکیل شده است که عبارت است از (همانطور که از شکل مقابل مشخص است):

- شبکه CNN برای استخراج ویژگی
- SVM classifier برای object detection
- مدل رگرسیونی برای تعیین کردن مختصات جعبه های محاطی

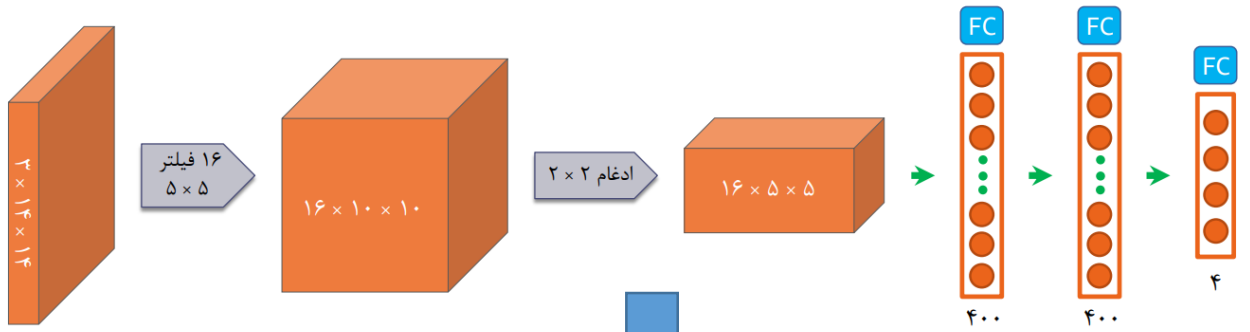


بنابراین این الگوریتم شناسایی Object بسیار کند است (زیرا هر یک از موارد بالا بسیار زمان بر است) بنابریان برای زمانی که می خواهیم الگوریتم به صورت Real-time کار کند نمی توانیم از این الگوریتم استفاده کنیم. در حال حاضر ورژن سریع تری از این الگوریتم وجود دارد faster RCNN و یا fast RCNN

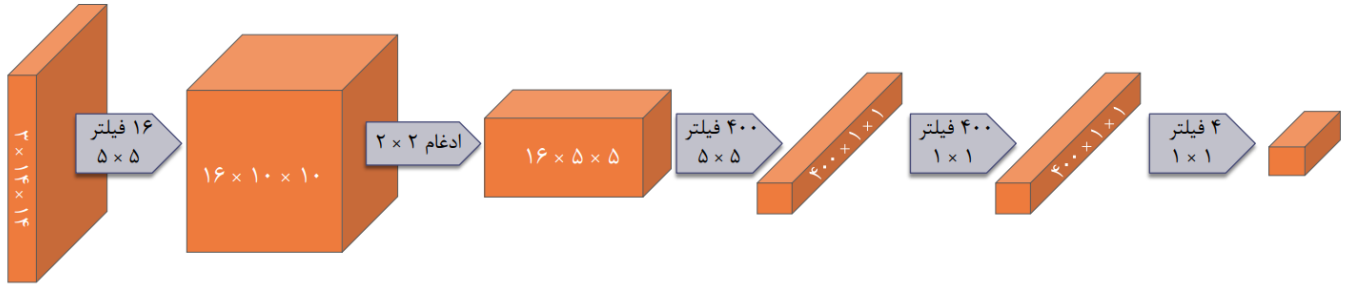
توضیحی در مورد R-FCN

برخلاف RCCN که از نظر محساباتی بسیار پیچیده و از نظر زماین بسیار کند است الگوریتم R-FCN بسیار بهینه و با دقت است. همانطور که اسم این الگوریتم مشخص است region based fully convolutional network، این الگوریتم از لایه های کاملا کانوولوشنی استفاده می کند. در واقع ما تمامی لایه های fully connect را نیز به صورت لایه های کانوولوشنی پیاده سازی می کنیم (زیرا لایه های کانوولوشنی سربار محاسباتی زیادی دارند) همانطور که در شکل زیر می بینیم:

شبکه ی کانوولوشنی با لایه های fully connected



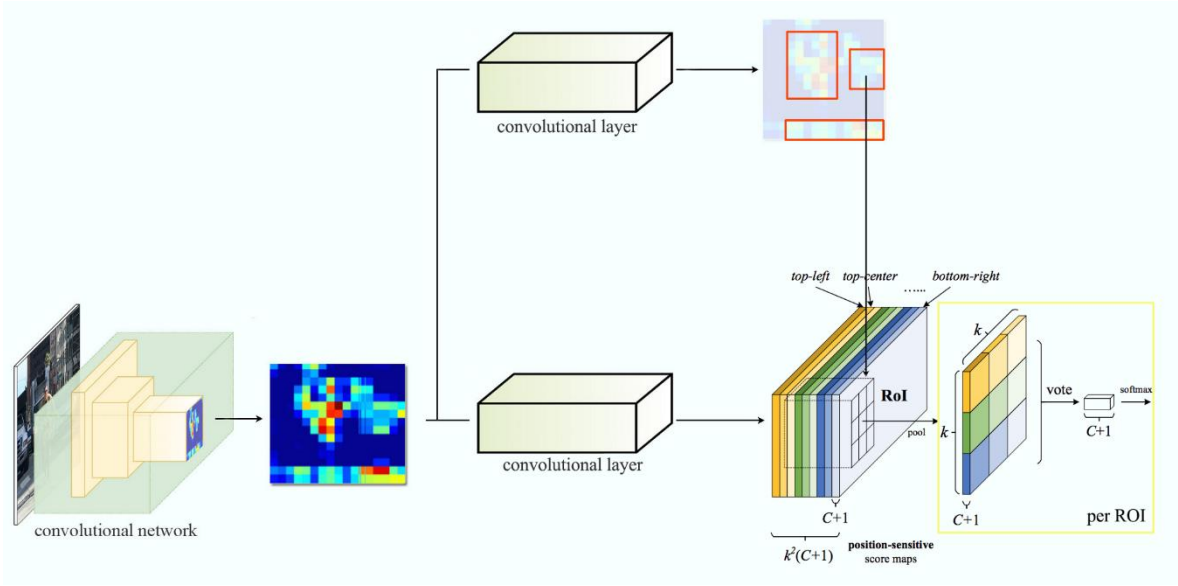
شبکه ی کانوولوشنی فقط با لا یه های کانوولوشنی



توضیحی در مورد R-FCN

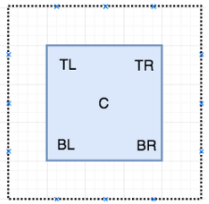
این الگوریتم به دنبال ایجاد یک مصالحه بین translation-invariance در مسائل دسته بندی و translation-variance در تشخیص تصاویر با استفاده از position-sensitive score map و position-sensitive ROI pooling است.

بر اساس این الگوریتم ما به جای استفاده از نواحی مختلف استفاده می کنیم همانطور که در شکل زیر می بینم با استفاده از یک شبکه ی کانوولوشنی و بر اساس یک region proposal network (RPN) شبیه RCNN، ROI های تصویر را استخراج می کنیم. که بخش های مختلف یک تصویر را پوشش می دهد. در واقع این الگوریتم به دنبال استفاده از امکان انجام محاسبات به صورت share هستیم و این مسئله به ما کمک می کند که بخش های مختلف یک تصویر را به صورت موازی و همزمان پردازش کنیم همانطور که در زیر میبینم:

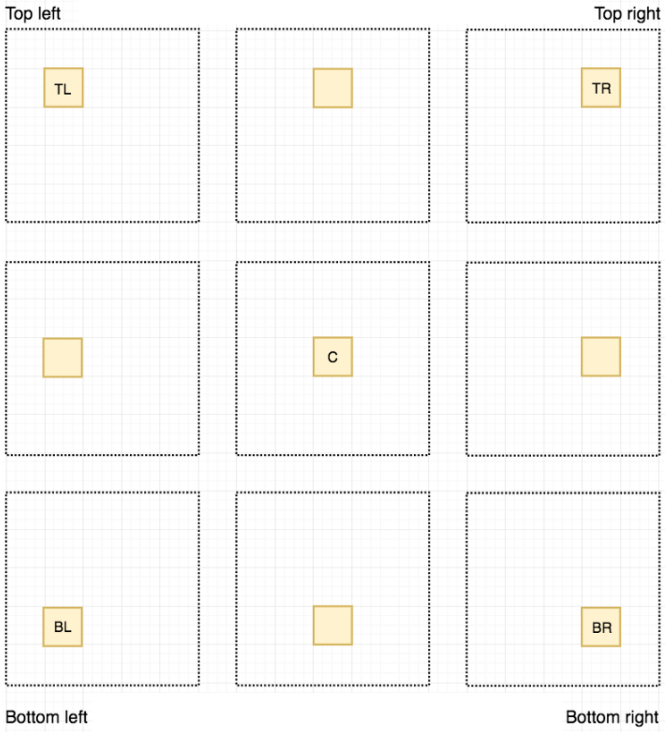


همانطور که در شکل زیر می بنیم علاوه بر لایه ی کانوولوشنی لایه کانولوشنی بر ای استحراج ROI یک لایه ی کانوولوشنی دیگر نیز وجود دارد که ویژگی های استخراج شده از آن توسط یک k grid در k به بخش های مختلف تقسیم می شود و احتمال وجود هر قسمت از ان Object در آن قسمت از gridمحاسبه می شود(score map) همانطور که در شکل زیر می بینم. در واقع به این ویژگی های اسخراج شده به ازای هر خانه از grid، position-sensitive score maps می گویند.بنابراین ما به ازای هر کلاس این k grid در k را داریم بنابراین کل ویژگی های استخراج شده برابر می شود با K*K(c+1) است که c تعداد کلاس ها و ۱ برای حالتی است که object وجود ندارد (background).

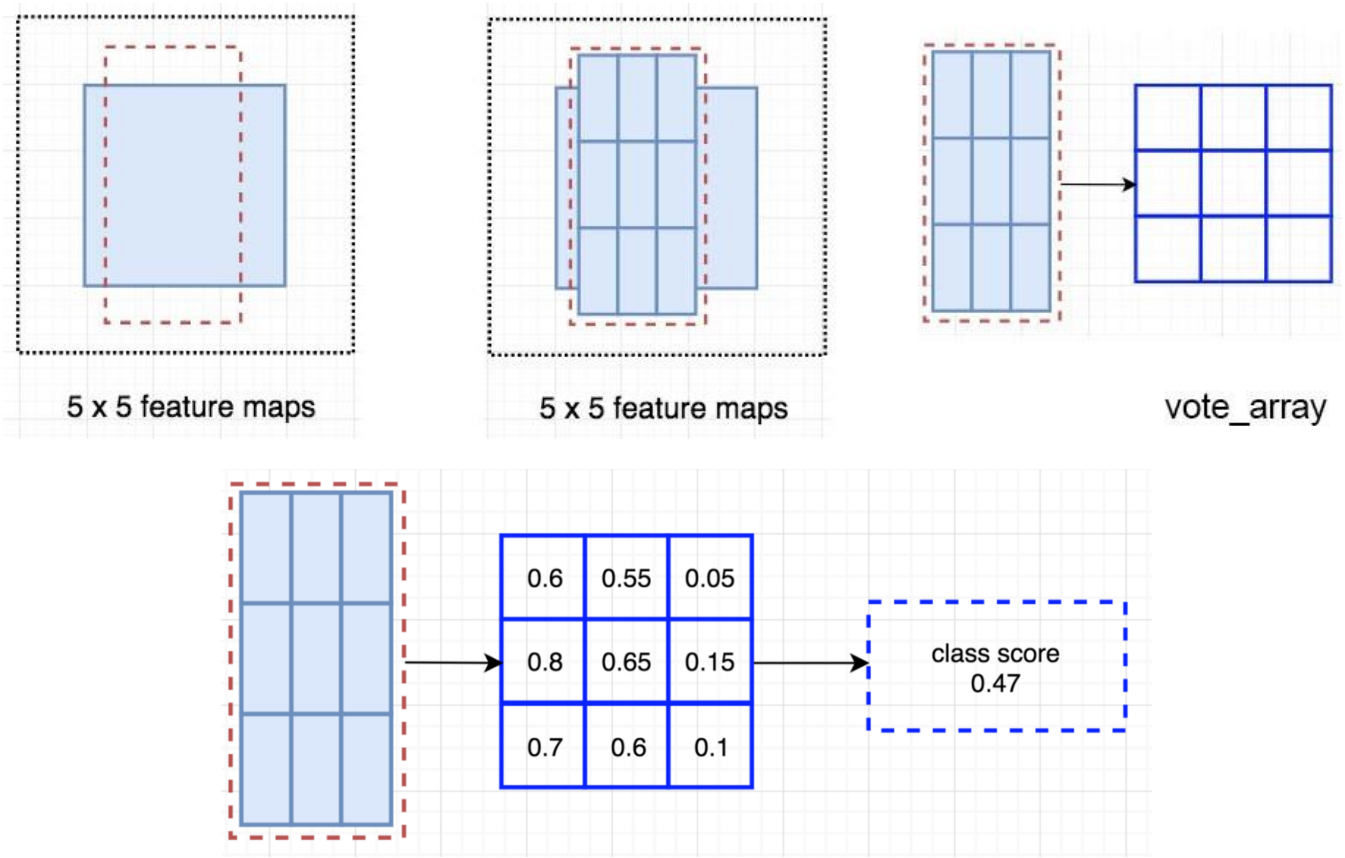
$$r(i,j \mid \theta) = \sum_{(x,y)\in bin(i,j)} z_{i,j,c}(x+x_0, y+y_0 \mid \theta)/n$$



5 x 5 feature maps



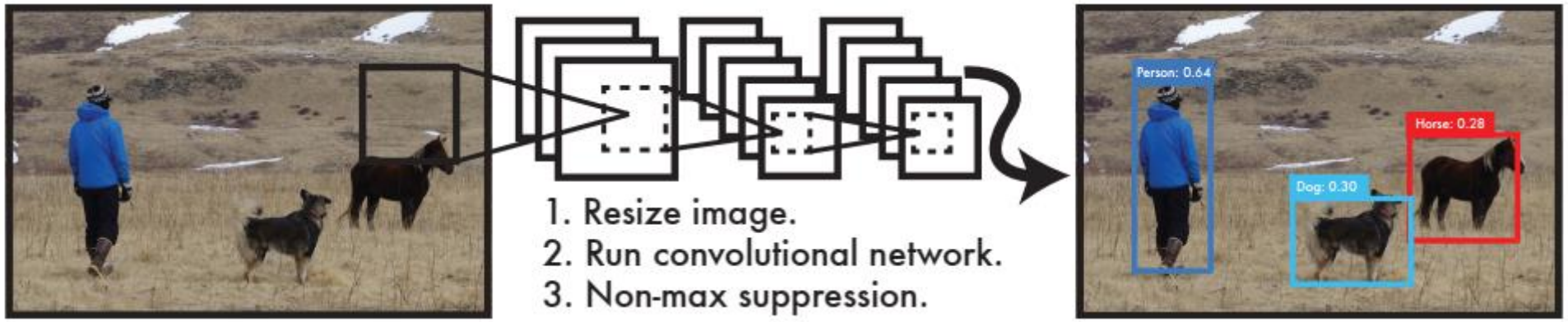
عملیات بعدی ترکیب ROI های بدست آمده با position-sensitive score maps هاست که به این مرحله position-sensitive ROI-pool می گویند. که به ازای هر class ما یک جدول vote خواهیم داشت که با استفاده از یک میانگین و سپس soft max احتمال هر کلاس را بدس می آوریم.

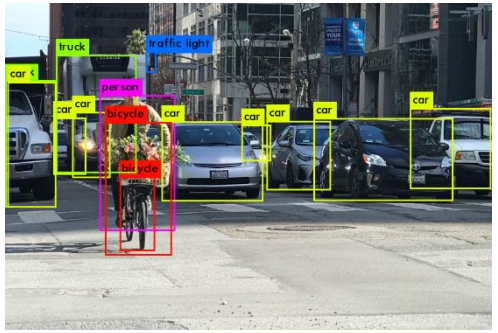


R-FCNبا کاهش میزان سربار مورد نیاز مورد نیاز برای هر ROIسرعت را بهبود می بخشد . region-based feature maps مستقل از ROIهستند و می‌توانند خارج از هر ROIمحاسبه شوند .بنابراین محسابات ساده تر است و بنابراین R-FCN سریعتر از سریعتر R-CNNاست.

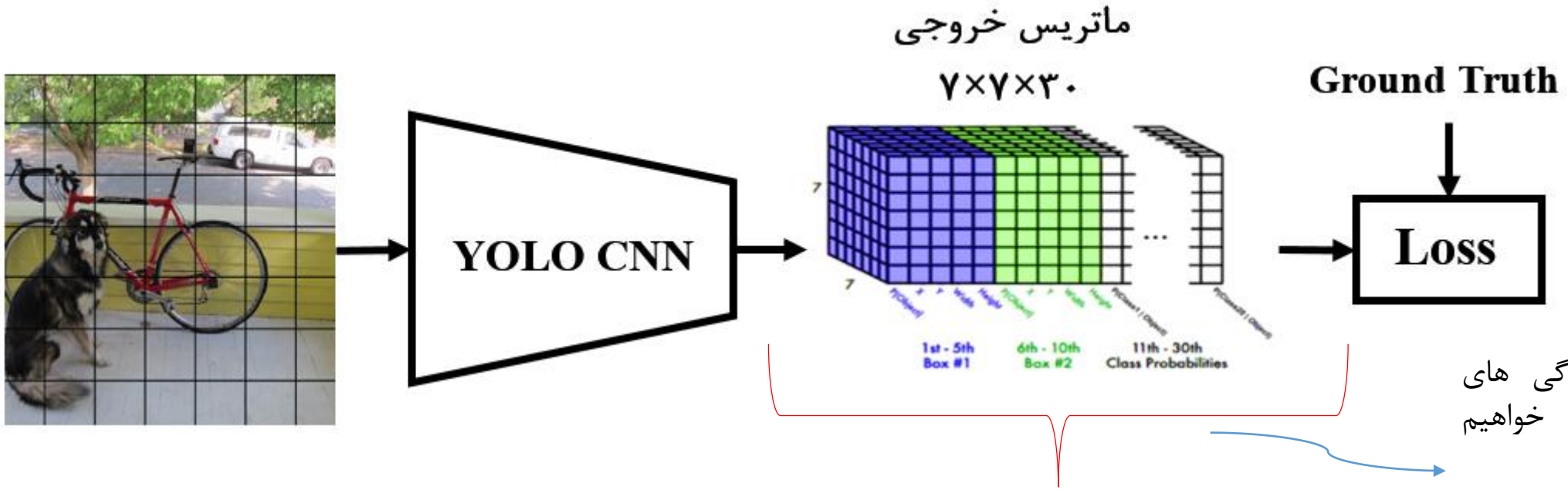
توضیحی در مورد YOLO

مخفف عبارت You Only Look Once، به‌معنای "شما فقط یک‌بار به تصویر نگاه می‌کنید" هست. درواقع، این عبارت به همان قابلیت سیستم بینایی انسان اشاره دارد که با یک نگاه عمل تشخیص اشیا را انجام می‌دهد. بر خلاف روش های موجود در تشخیص اشیا مثل RCNN و RFCN الگوریتم YOLO مسئله ی تشخیص تصاویر را مثل یک مسئله ی رگرسیون در نظر میگیرد و مستقیما از پیکسل‌های تصویر به مختصات باکس و احتمال کلاس‌ها می‌رسد. همانطور که در شکل زیر نگاه می کنیم.





درالگوریم YOLO تصویر ورودی با ابعاد ۳×۴۴۸×۴۴۸ به یک Gridیا شبکهS×S تقسیم بندی می‌شود. این تصویر به شبکه YOLOداده می‌شود. خروجی شبکه کانولوشنی، ماتریسی به ابعاد ۳۰× S×S خواهد بود. هریک از درایه‌های ماتریس S×S خروجی معادل با یک سلول در شبکه S×S ورودی است (به ورودی و خروجی در شکل ۵ دقت کنید). خروجی ۳۰× S×Sشامل مختصات باکس‌ها و احتمال‌هاست. اگر در فرآیند آموزش Trainباشیم، خروجی ۳۰× S×S به‌همراه باکس‌های واقعی (Ground Truth) به تابع هزینه داده می‌شود. مقدار S در یولو نسخه ۱، برابر با ۷ درنظر گرفته شده است. اگر در فرآیند آزمایشTest باشیم، خروجی ۳۰× S×S به الگوریتم حذف غیرحداکثرها Non-maximum Suppression داده می‌شود (برای این که ممکن است چندین باکس به صورت همپوشانی شده روی یک شی قرار بگیرند) تا باکس‌های ضعیف از بین بروند و تنها باکس‌های درست در خروجی نمایش داده شوند. درادامه، درمورد طراحی شبکه YOLO، نحوه آموزش شبکه، تابع هزینه، آزمایش شبکه و غیره توضیح خواهیم داد.



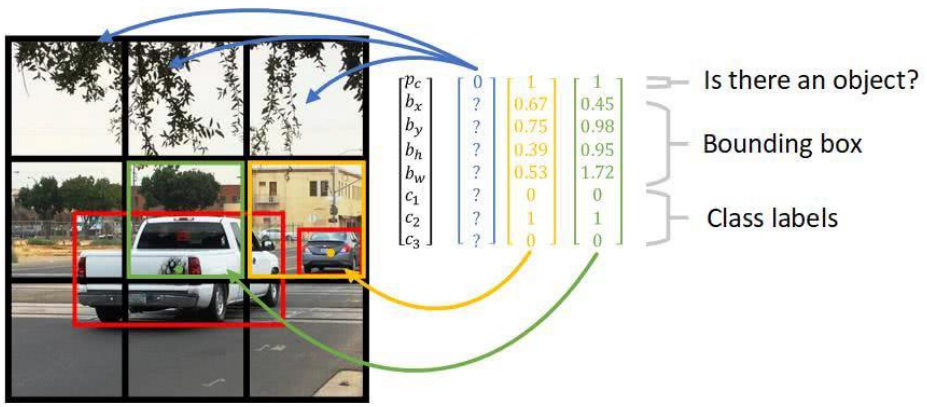
در ادامه به ویژگی های استخراج شده خواهیم پرداخت

در الگوریتم YOLO ما به ازای هر قسمت از تصویر که Grid بندی شده است یک سری بردار به عنوان Y در نظر می گیریم که به صورت زیر تعریف می شود.

$$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ C_1 \\ C_2 \\ C_3 \end{bmatrix}$$

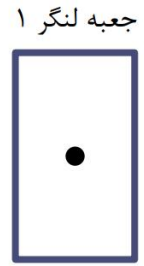
همانطور که گفته شد YOLO مثل یک مدل رگرسیونی عمل می کند بنابریان به ازای هر قسمت از grid این برداری را محاسبه می کند که شامل احتمال حضور یک object در آن قسمت است p. مختصات جعبه محاطی bx , by, bh, bw و احتمال تعلق به هرکلاس c1 تا cn

همچنین در این الگوریتم از یک ویژگی به اسم لَنگرد استفاده می شود که این لنگر ها به نوعی میانگینی از ابعاد باکس متناظر برای object های متفاوت است. مثلا لنگر مربوط به اتوبوس باکسی شبیه مستطیل دارد. به ازای لننگر های مختلف ما برداری شبیه بردار بالا را تعریف می کنیم.

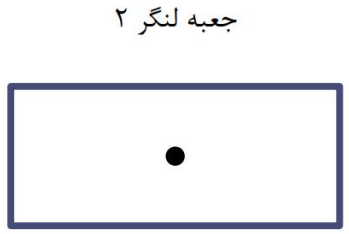


$$y = \begin{bmatrix} y_{a1} \\ y_{a2} \end{bmatrix}$$

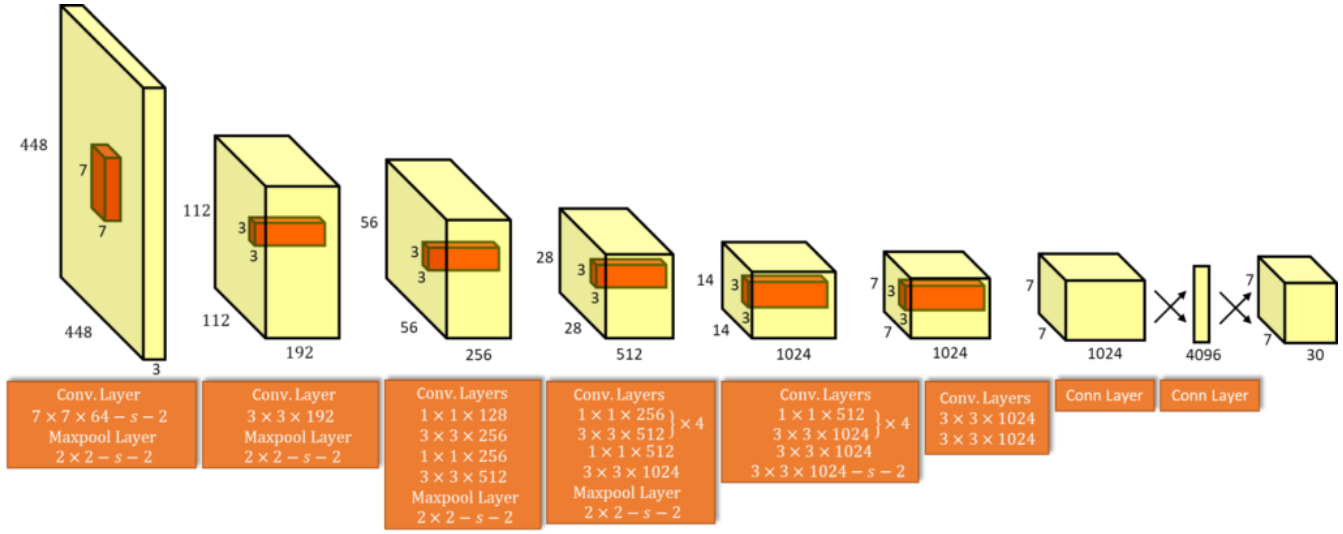
به ازای هر خانه



$$y_{a1} = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ C_1 \\ C_2 \\ C_3 \end{bmatrix}$$



$$y_{a2} = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ C_1 \\ C_2 \\ C_3 \end{bmatrix}$$

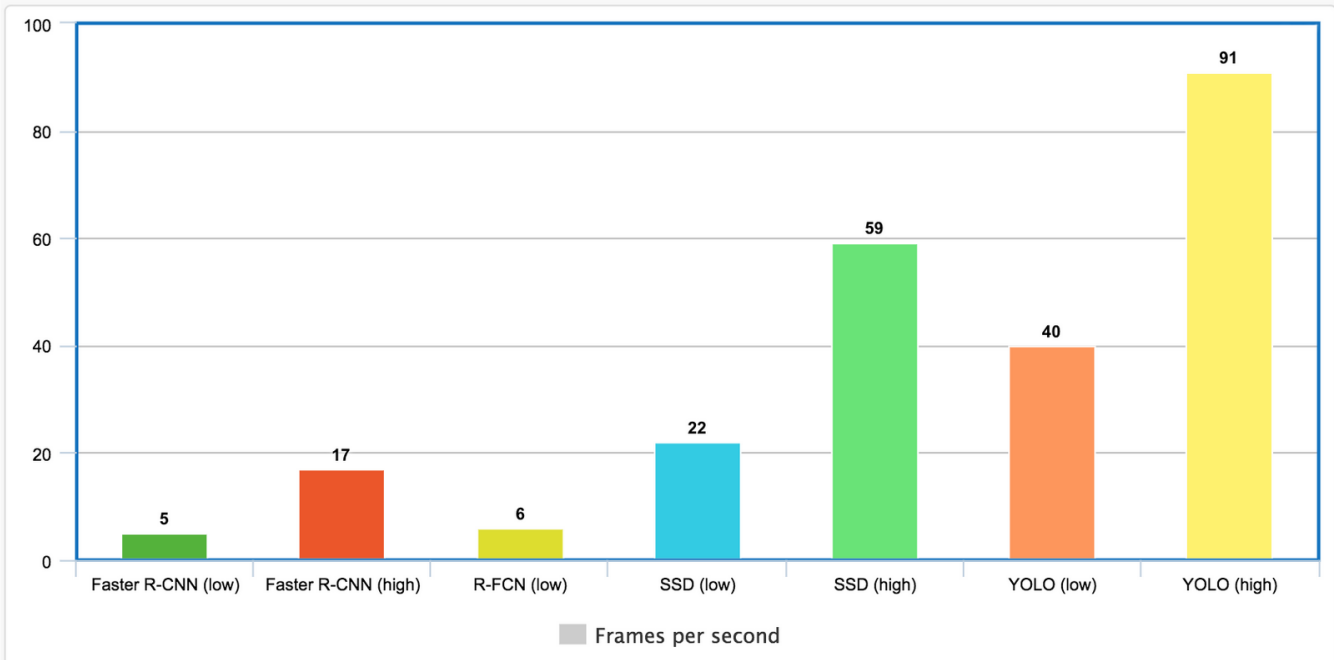


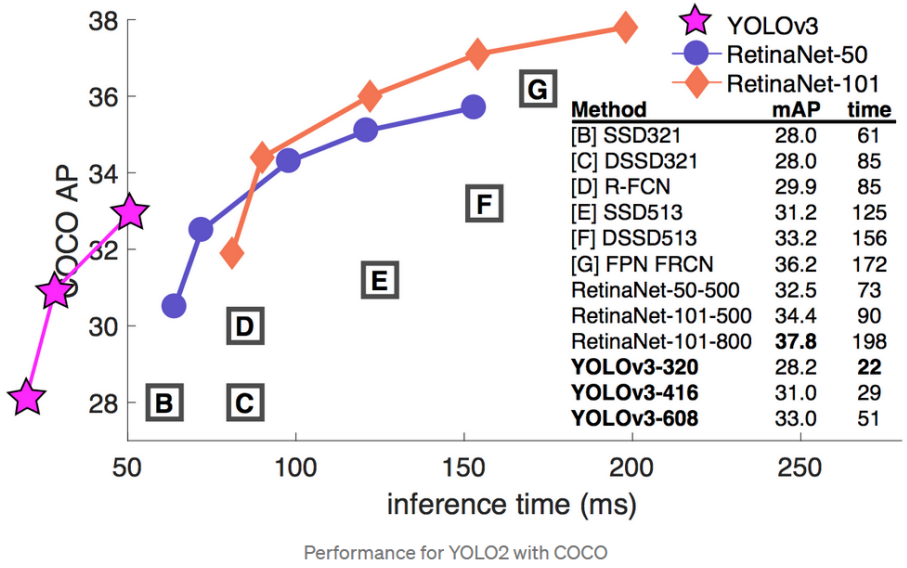
شامل یک شبکه عصبی کانولوشنی Convolutional Neural Network با ۲۴ لایه کانولوشنی برای استخراج ویژگی و همچنین ۲ لایه فولی‌کانکتد Fully Connectedبرای پیش‌بینی احتمال و مختصات اشیا است.

همچنین، یک نسخه سریع از YOLO برای جابجایی مرزهای تشخیص اشیای سریع طراحی شده است. YOLO سریع، یک شبکه عصبی با تعداد لایه‌های کانولوشنی کمتر است که در آن از ۹ لایه کانولوشنی بجای ۲۴ لایه کانولوشنی (YOLO اصلی) استفاده شده و البته تعداد فیلترهای هر لایه در YOLO سریع نسبت به YOLO اصلی کمتر است. اندازه ورودی هر دو شبکه ۳×۴۴۸×۴۴۸ و خروجی شبکه نیز یک تنسور ۳۰×۷×۷ از پیش‌بینی‌ها است. درتمامی لایه‌ها از Leaky ReLU استفاده شده است

بسیار سریع است. در اینجا، تنها یک شبکه وجود دارد که خیلی ساده به آن ورودی تصویر داده می‌شود (برخلاف RCNN و RFCN) تا شبکه پیش‌بینی‌های تشخیص اشیا را به ما نشان دهد. دو نسخه شبکه YOLOشامل YOLOاصلی و YOLO سریع طراحی شده است.

بنابراین با توجه به نکات گفته شده الگوریتم YOLO برای اپلیکیشن های Real-time بسیار مناسب تر از الگوریتم های دیگر است. نمونه ای مقایسه ای سرعت YOLO با دیگر الگوریتم ها در شکل زیر وجود دارد.





Performance for YOLO2 with COCO

یولو اصلی با کارت گرافیک Titan X با سرعت ۴۵ فریم برثانیه اجرا می‌شود. نسخه سریع YOLO هم سرعتی بیش از ۱۵۰ فریم‌برثانیه دارد. یعنی YOLO می‌تواند در یک ویدئوی ۴۰ فریم برثانیه درحالت بلادرنگ به تشخیص اشیا بپردازد. YOLO نسبت به دیگر سیستم‌های تشخیص اشیای بلادرنگ، به mAP یا همان mean Average Precision دوبرابر دست یافته است. عملکرد بهتر نسبت به سایر سیستم‌های بلادرنگ و نه سیستم‌های تشخیص اشیای قدرتمند مانند Faster R-CNN که بلادرنگ نیستند.

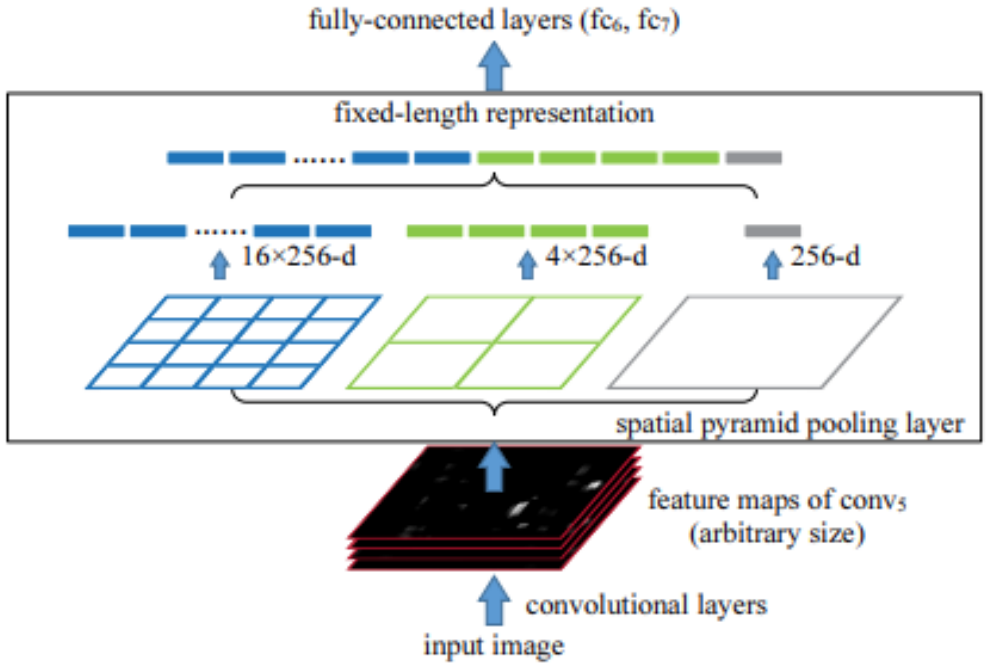
$$\lambda_{coord}\sum_{i=0}^{s^2}\sum_{j=0}^{B}1_{ij}^{obj}[(x_i-\hat{x}_i)^2+(y_i-\hat{y}_i)^2]+$$

$$\lambda_{coord}\sum_{i=0}^{s^2}\sum_{j=0}^{B}1_{ij}^{obj}[(\sqrt{w_i}-\sqrt{\hat{w}_i})^2+(\sqrt{h_i}-\sqrt{\hat{h}_i})^2]+$$

$$\sum_{i=0}^{s^2}\sum_{j=0}^{B}1_{ij}^{obj}(C_i-\hat{C}_i)^2+\lambda_{coord}\sum_{i=0}^{s^2}\sum_{j=0}^{B}1_{ij}^{obj}(C_i-\hat{C}_i)^2+$$

$$\sum_{i=0}^{s^2}1_{ij}^{obj}\sum_{c\in classes}(p_i(c)-\hat{p}_i(c))^2$$

❖ زمانی که ما از شبکه های کانوولوشنی برای استخراج ویژگی و از لایه های fully connected برای دسته بندی و پیش بنی استفاده می کنیم به ناچار مجبور می شویم که ابعاد ویژگی های ورودی شبکه ی کانوولوشنی را یکسان در نظر بگیریم و این مسئله به نوعی برای ما محدودیت ایجاد می کند زیرا لایه ی fully connect در غیر اینطورت دچار مشکل می شد. بنابراین گاهی برای اینکه ابعاد ویژگی های ورودی را یکسان کنیم مجبور می شویم داده ها را reshape کنیم که خود این مسئله می تواند ورودی های شبکه و خروجی آن را تحت تاثیر قرار دهد. بنابریان ایده ی استفاده از Spatial Pyramid pooling شکل گرفت. طبق این فرضیه وقتی ما لایه های ورودی را از لایه های کانوولوشنی عبور میدادیم feature map هایی را استخراج میکنیم می توانیم لایه های پولینگ (مثلا ۳ لایه) را در نظر بگیریم و با استفاده از آن ابعاد را یکسان کنیم همانطور که در شکل زیر می بینیم.



همانطور که در شکل مقابل می بینیم پس از استخراج ویژگی ها از لایه های کانوولوشی ما یک سری ویژگی با عمیق d داریم به ازای هر لایه (هر bin) مثلا ما یک لایه پولینگ با سایز ۱ در نظر میگیریم بنابر این به ازای هر bin و بر اساس متد pooling که در نظر گرفتیم مثلا max pooling یک ویژگی استخراج می کنیم بنابر این پس از اولین pooling ما به اندازه ی d تا ویژگی داریم. سپس یک لایه pooling دیگر با سایز دو مثلا ایجاد می کنیم که در این صورت به ازای هر bin ما ۴ تا ویژگی استخراج می کنیم که در این صورت کل ویژگی های استخارج شده برابر با 4*d می شود. پس از این لایه یک لایه ی پولینگ دیگر استفاده می کنیم با سایز پنچره ۴ بنابریان در هر bin ما ۱۶ ویژگی استخراج می کنیم که در این صورت کل ویژگی های استخراج شده در این لایه برابر با 16*d می شود. بنابراین کل ویژگی های استخراج شده به ازای لایه ی Spatial Pyramid pooling برابر با 16d+4d+d می شود که یک سایز ثابت است.

در مسئله ی object detection ما نیاز داریم تا بخش های تصویر را crop کنیم یا wrap کنیم و تعداد زیادی تصویر با ابعاد مختلف ایجاد می شود (بر اساس RCNN). بنابراین پیوسته ابعاد تصویر ورودی ما متغیر هستند و روش Spatial Pyramid pooling می تواند بسیار کمک کننده باشد. بر این اساس به ازای هر تصویر با سایز گوناگون لایه ی ssp خروجی با سایز یکسان تولید می کند همانطور که در شکل زیر می بینیم: