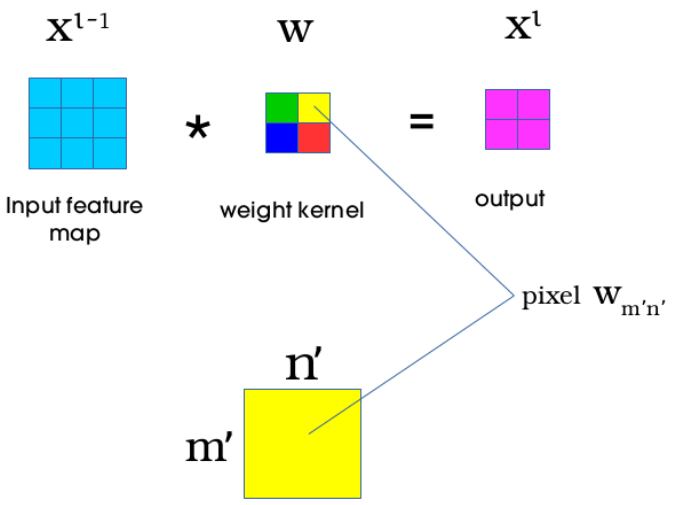
- **سوال 1:**

توضیحات مربوط به Backpropagation، طبق منبع اعلامی در صورت سوال می باشد، پس مثالی هم که در این بخش از آموزش شبکه را توضیح می دهم نیز بر اساس مثال همان منبع می باشد. همچنین طبق Notation توضیح داده شده در لینک به بحث در مورد Backpropagation پرداخته می شود.

به طور کلی فرایند Backpropagation در شبکه های کانولوشنال به دو فاز محاسبه مشتقات جزئی و اعمال Chain Rule و پس از آن به روز رسانی پارامتر های شبکه تقسیم می شود. این به روز رسانی شامل پارامترهای وزن و دلتا می شود.

- **وزن ها:**

در این جا ما به دنبال محاسبه تاثیر تغییرات وزن یک پیکسل از لایه های شبکه بر مقدار تابع هزینه می باشد. یعنی به دنبال محاسبه $\frac{\partial E}{\partial w^{l}_{m',n'}}$ که منظور از E، همان تابع هزینه و منظور از $w_{m',n'}$ ، یک پیکسل از کرنل وزن شبکه می باشد.



با توجه شکل بالا، در بخش Forward Propagation، از آن جا که ماتریس Feature Map در ماتریس Weight Kernel ضرب شده اند و پیکسل زرد رنگ $w_{m',n'}$ در همه ضرب های صورت گرفته دخیل بوده، پس در نتیجه این پیکسل مانند پیکسل های دیگه در خروجی به دست آمده تاثیر گذار می باشد.

در واقع بعد ماتریس Feature map خروجی به روش زیر محاسبه می شود:

ابعاد ماتریس weight kernel، $k_1 \times k_2$ ابعاد feature map ورودی، که با ضرب انجام شده به ابعاد زیر در مورد ماتریس feature map خروجی میرسیم: $H \times W$

$$(H - k_1 + 1) \text{ by } (W - k_2 + 1)$$

حال با توجه به ابعاد ماتریس های خروجی و وزن، رابطه $\frac{\partial E}{\partial w^l_{m',n'}}$ به فرمت زیر تبدیل می شود:

$$\begin{aligned}\frac{\partial E}{\partial w^l_{m',n'}} &= \sum_{i=0}^{H-k_1}\sum_{j=0}^{W-k_2} \frac{\partial E}{\partial x^l_{i,j}} \frac{\partial x^l_{i,j}}{\partial w^l_{m',n'}} \\ &= \sum_{i=0}^{H-k_1}\sum_{j=0}^{W-k_2} \delta^l_{i,j} \frac{\partial x^l_{i,j}}{\partial w^l_{m',n'}} \end{aligned} \tag{10}$$

که منظور از $x^l_{i,j}$ در رابطه بالا، $\sum_m \sum_n w^l_{m,n} o^{l-1}_{i+m,j+n} + b^l$ می باشد که با جایگذاری و بسط دادن سیگما ها و همچنین با تغییر بعضی علائم مانند $m = m' \text{ and } n = n'$ ، تبدیل به رابطه زیر می شود:

$$\begin{aligned}\frac{\partial x^l_{i,j}}{\partial w^l_{m',n'}} &= \frac{\partial}{\partial w^l_{m',n'}} \left(w^l_{0,0} o^{l-1}_{i+0,j+0} + \cdots + w^l_{m',n'} o^{l-1}_{i+m',j+n'} + \cdots + b^l \right) \\ &= \frac{\partial}{\partial w^l_{m',n'}} \left(w^l_{m',n'} o^{l-1}_{i+m',j+n'} \right) \\ &= o^{l-1}_{i+m',j+n'} \end{aligned} \tag{12}$$

با جایگذاری معادله 12 در معادله 10 داریم:

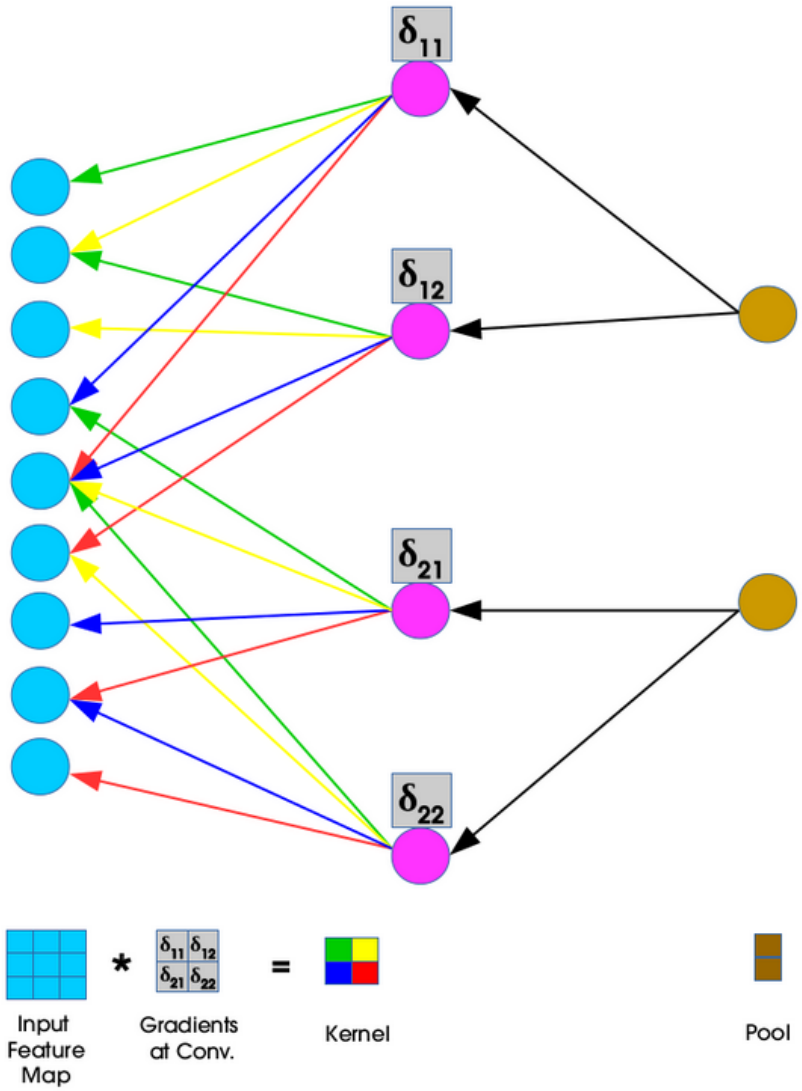
$$\frac{\partial E}{\partial w^l_{m',n'}} = \sum_{i=0}^{H-k_1}\sum_{j=0}^{W-k_2} \delta^l_{i,j} o^{l-1}_{i+m',j+n'} \tag{13}$$

که به عبارتی، می توان معادله بالا رو به فرمت زیر هم نوشت:

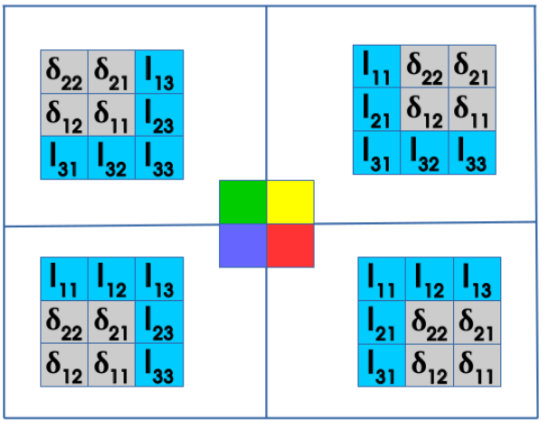
$$= \mathrm{rot}_{180^\circ} \left\{ \delta_{i,j}^{l} \right\} * o_{m',n'}^{l-1} \qquad (14)$$

زیرا جمع صورت گرفته نشان دهنده همه گرادیان های خروجی $\delta_{i,j}^{l}$ در لایه L می باشد.
در واقع اگر با توجه به filter map در فاز forward به این گرادیان ها نگاه کنیم، با انجام یک عملیات flipping می توان یک همبستگی بین ماتریس گرادیان ها و filter map بیان کرد. به عبارتی با توجه به توضیحات بالا، منظور از $\mathrm{rot}_{180^\circ} \left\{ \delta_{i,j}^{l} \right\}$ در معادله 14، شکل زیر می باشد:

و همچنین طریقه به وجود آمدن ماتریس دلتا ها به این فرمت در شبکه در فاز Backpropagation، در شکل زیر قابل درک می شود:



در واقع ماتریس دلتا در عملیات کانولوشن به روش زیر باعث به روز رسانی وزن های kernel weight می شود:

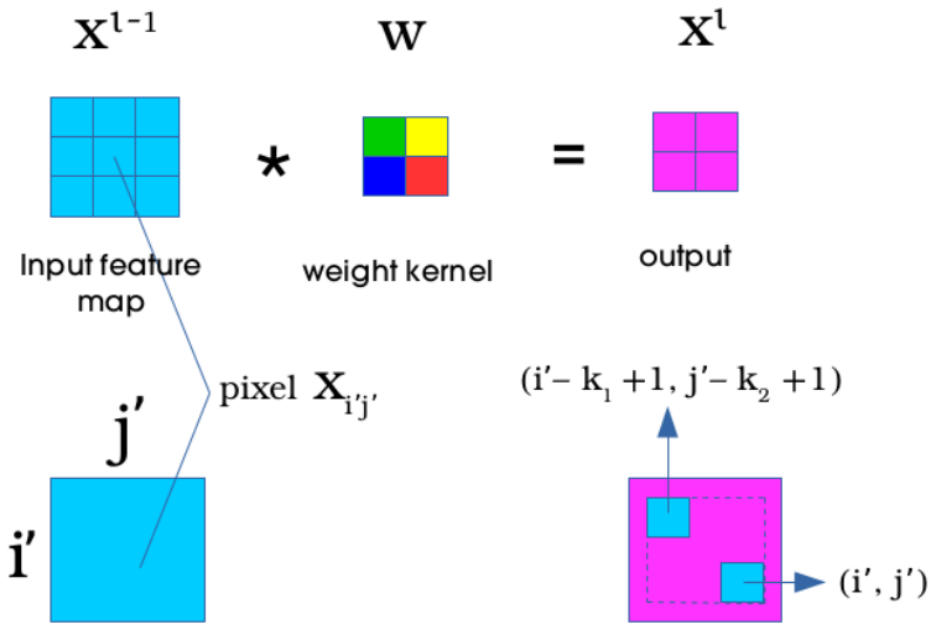


- **دلتا ها:**

دقت کنید که تمام دلتا های مورد استفاده در ساخت وزن های جدید از تاثیر تغییرات feature map ورودی بر تابع هزینه به وجود می آیند. یعنی به عنوان مثال برای لایه L داریم:

$$\delta_{i,j}^{l} = \frac{\partial E}{\partial x_{i,j}^{l}} \qquad (15)$$

که مانند محاسبه تاثیر تغییرات وزن های kernel weight بر تابع هزینه، می توان نشان داد که منظور از دلتا و معادله 15 چیست:



در این جا هم مانند بخش مربوط به وزن ها، به محاسبه مقدار مشتق جزئی $\frac{\partial E}{\partial x_{i,j}^{l}}$ می پردازیم:

$$\begin{aligned}\frac{\partial E}{\partial x_{i',j'}^{l}} &= \sum_{m=0}^{k_1-1}\sum_{n=0}^{k_2-1}\frac{\partial E}{\partial x_{i'-m,j'-n}^{l+1}}\frac{\partial x_{i'-m,j'-n}^{l+1}}{\partial x_{i',j'}^{l}} \\ &= \sum_{m=0}^{k_1-1}\sum_{n=0}^{k_2-1}\delta_{i'-m,j'-n}^{l+1}\frac{\partial x_{i'-m,j'-n}^{l+1}}{\partial x_{i',j'}^{l}} \end{aligned} \tag{17}$$

که در این معادله مانند معادله 10، منظور از $x_{i'-m,j'-n}^{l+1}$ همان:

$$\sum_{m'}\sum_{n'} w_{m',n'}^{l+1} o_{i'-m+m',j'-n+n'}^{l} + b^{l+1}$$

که با بسط دادن آن و محاسبه سیگما ها داریم:

$$\frac{\partial x_{i'-m,j'-n}^{l+1}}{\partial x_{i',j'}^{l}} = \frac{\partial}{\partial x_{i',j'}^{l}} \left(\sum_{m'} \sum_{n'} w_{m',n'}^{l+1} o_{i'-m+m',j'-n+n'}^{l} + b^{l+1} \right)$$
$$= \frac{\partial}{\partial x_{i',j'}^{l}} \left(\sum_{m'} \sum_{n'} w_{m',n'}^{l+1} f\left(x_{i'-m+m',j'-n+n'}^{l} \right) + b^{l+1} \right) \quad (18)$$

در اینجا با تغییر $n = n'$ و $m = m'$ ، $f\left(x_{i'-m+m',j'-n+n'}^{l} \right)$ را به $f\left(x_{i',j'}^{l} \right)$ و همچنین $w_{m',n'}^{l+1}$ را به $w_{m,n}^{l+1}$ تبدیل می کنیم. در نتیجه، معادله 18 تبدیل به معادله زیر می شود:

$$\frac{\partial x_{i'-m,j'-n}^{l+1}}{\partial x_{i',j'}^{l}} = \frac{\partial}{\partial x_{i',j'}^{l}} \left(w_{m',n'}^{l+1} f\left(x_{0-m+m',0-n+n'}^{l} \right) + \cdots + w_{m,n}^{l+1} f\left(x_{i',j'}^{l} \right) + \cdots + b^{l+1} \right)$$
$$= \frac{\partial}{\partial x_{i',j'}^{l}} \left(w_{m,n}^{l+1} f\left(x_{i',j'}^{l} \right) \right)$$
$$= w_{m,n}^{l+1} \frac{\partial}{\partial x_{i',j'}^{l}} \left(f\left(x_{i',j'}^{l} \right) \right)$$
$$= w_{m,n}^{l+1} f'\left(x_{i',j'}^{l} \right) \quad (19)$$

و با جایگذاری آن در معادله 17 و انجام ساده سازی ها، داریم:

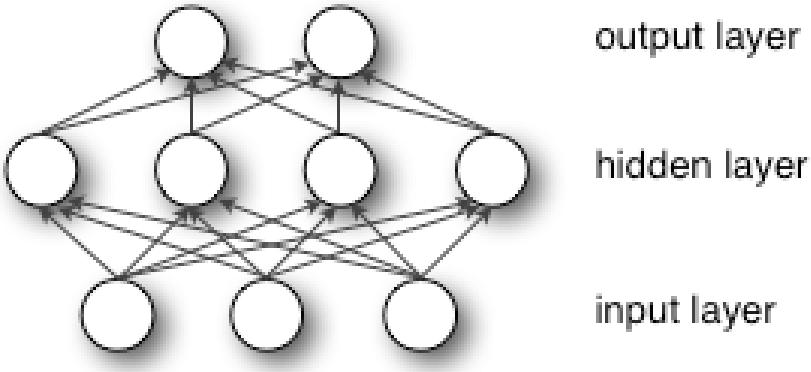
$$\frac{\partial E}{\partial x_{i',j'}^{l}} = \sum_{m=0}^{k_1-1} \sum_{n=0}^{k_2-1} \delta_{i'-m,j'-n}^{l+1} w_{m,n}^{l+1} f'\left(x_{i',j'}^{l} \right) \quad (20)$$

برای انتشار معکوس(Backpropagation)، از هسته برگشتی(Flipped Kernel) استفاده می کنیم و در نتیجه یک پیچیدگی یا کانولوشن خواهیم داشت که به صورت همبستگی متقاطع با یک هسته برگشتی بیان می شود:

$$\frac{\partial E}{\partial x_{i',j'}^{l}} = \sum_{m=0}^{k_1-1} \sum_{n=0}^{k_2-1} \delta_{i'-m,j'-n}^{l+1} w_{m,n}^{l+1} f'\left(x_{i',j'}^{l} \right)$$
$$= \mathrm{rot}_{180^\circ} \left\{ \sum_{m=0}^{k_1-1} \sum_{n=0}^{k_2-1} \delta_{i'+m,j'+n}^{l+1} w_{m,n}^{l+1} \right\} f'\left(x_{i',j'}^{l} \right) \quad (21)$$

● سوال 2:

شکست تقارن یا Symmetry Breaking به طریقه وزن دهی اولیه شبکه های عصبی اشاره دارد.
شبکه عصبی که در ابتدا مقدار وزن هایش برابر باشد، ممکن است تغییر این وزن‌ها در حین آموزش مدل دشوار یا غیرممکن باشد. این همان مشکل "تقارن یا Symmetry" است. برای جلوگیری از این رخداد، می توان در ابتدا به مدل وزن های تصادفی نسبت داد.
به مثال زیر توجه کنید:



شکستن تقارن در اینجا ضروری است. در طول انتشار رو به جلو (Forward Propagation)، هر واحد در لایه پنهان مقدار زیر را دریافت می کند:

$$a_i = \sum_{i}^{N} W_{i,j} \cdot x_i$$

یعنی هر واحد پنهان مجموع ورودی ها را در وزن مربوطه ضرب می کند. حالا تصور کنید که همه وزن ها را به یک مقدار (مثلاً صفر یا یک) مقداردهی اولیه کنید. در این حالت، هر واحد پنهان دقیقاً همان سیگنال را دریافت می کند. به عنوان مثال، اگر همه وزن ها به عدد 1 مقداردهی شوند، هر واحد سیگنالی برابر با مجموع ورودی ها دریافت می کند. اگر همه وزن ها صفر باشند، که این حتی بدتر از مقدار 1 است، هر واحد پنهان سیگنال صفر دریافت می کند. مهم نیست که ورودی چه بوده است - اگر همه وزن ها یکسان باشند، همه واحدهای لایه پنهان نیز یکسان خواهند بود.

این مشکل اصلی در مورد تقارن و دلیلی است که چرا باید وزن ها را به طور تصادفی (یا حداقل با مقادیر مختلف) مقداردهی اولیه کنیم. توجه داشته باشید که این مشکل بر تمام معماری هایی که از اتصالات each-to-each استفاده می کنند تأثیر می گذارد.

- **سوال 3:**

به طور کلی در این بخش به توضیح مزایا و معایب لایه های ادغام یا Pooling Layers می پردازیم.

- مزایا:
  - لایه های ادغام (Pooling Layers)برای کاهش ابعاد نقشه های ویژگی (Feature Maps) استفاده می شود. بنابراین، تعداد پارامترهای یادگیری و میزان محاسبات انجام شده در شبکه را کاهش می دهد.
  - لایه ادغام، ویژگی های موجود در یک منطقه از نقشه ویژگی ایجاد شده توسط یک لایه کانولوشن را خلاصه می کند. بنابراین، عملیات بیشتری بر روی ویژگی‌های خلاصه شده به جای ویژگی‌های دقیقاً موقعیت‌یافته تولید شده توسط لایه پیچیدگی انجام می‌شود. این باعث می شود که مدل نسبت به تغییرات در موقعیت ویژگی ها در تصویر ورودی قوی تر باشد.
- معایب:
  - برای مثال در Max pooling، حداکثر گسسته را در شبکه پیکسل انتخاب می کنیم که بسیاری از اوقات حداکثر واقعی نیست. اگر حداکثر ویژگی دیتای شما در میان حداکثر ویژگی ای که توسط max pooling انتخاب شده، نباشد، در نهایت مدل شما با ویژگی ها نادرستی آموزش خواهد دید.
  - در Average pooling هم که اگر توزیع مقادیر در ماتریس تصویر یکسان نباشد یا دیتا دارای نویز باشد، ادغام میانگینی که انجام می شود، دقیق نخواهد بود.

در نهایت از آن جا که این لایه ها باعث کاهش ابعاد دیتا می شود و در نتیجه آن در مدت کم تر با محاسبات و مصرف کمتر منابع به دقت خوبی از مدل سازی رسید، استفاده از این لایه ها رو به صرفه تر از استفاده نکردن شون میکند.

در انتها هم افزودن یک لایه ادغام بعد از لایه کانولوشنال، یک الگوی رایج است که

برای مرتب کردن لایه‌ها در یک شبکه عصبی کانولوشن استفاده می‌شود که ممکن است یک یا چند بار در یک مدل مشخص تکرار شود.

- **سوال 4:**
  - **Quadratic loss:**

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

تابع Quadratic loss یا Mean Squared Error loss، در حقیقت به محاسبه میزان فاصله جواب خروجی مدل با مقدار واقعی آن می پردازد. علاوه بر آن، مسئله طبقه بندی، یک مسئله گسسته و غیر محدب است. به این معنی که تابع هزینه ای مانند MSE به علت عدم تحدب مسئله، نمی تواند به پاسخ درست مسئله برسد. چرا که با اعمال Gradient Descent تنها به نقطه اکسترمم محلی برای minimize کردن تابع هزینه میرسیم. اما این تابع هزینه برای مسائل پیوسته ای مثل Regression مناسب می باشد.

  - **Cross-Entropy**

$$L = -\frac{1}{m}\sum_{i=1}^{m} y_i \cdot \log(\hat{y}_i)$$

در تابع Cross-Entropy، آنتروپی معیار تصادفی بودن اطلاعات در حال پردازش است و Cross-Entropy معیار اندازه گیری تفاوت تصادفی بین 2 متغیر تصادفی می باشد. به دلیل عدم تحدب در بحث طبقه بندی(به علت وجود 2 یا چند کلاس گسسته) این

تابع برای بررسی جواب مدل و طبقه بندی آن به کلاس های موجود با محاسبه Maximum Likelihood، مناسب برای مسائل طبقه بندی می باشد.

بنابراین مهم ترین تفاوت میان Quadratic Loss & Cross-Entropy Loss، نوع خروجی و تحدب توابع هزینه می باشد که Cross-Entropy را برای با مسائل Classification و Quadratic Loss رو برای مسائل Regression مناسب می سازد.

- **سوال 5:**

Leaky RELU بهبود یافته ی تابع فعال ساز RELU می باشد. در واقع اگر رابطه هر دو تابع را بنویسیم، داریم:

$$f(x) = max(0, x) = \begin{cases} x & x \geq 0 \\ 0 & x < 0 \end{cases}$$

برای تابع RELU؛
و برای تابع Leaky RELU خواهیم داشت:

$$f(x) = max(0, x) = \begin{cases} x & x > 0 \\ 0 & mx \leq 0, \text{where } m \text{ is a preselected slope value} \end{cases}$$

طبق روابط بالا، تابع Leaky RELU دو مزیت اساسی دارد:

1. از آن جا که شیب مقادیر منفی آن صفر نمی باشد، می تواند مشکل "Dying RELU" رو برطرف کند.

***Dying RELU: این مشکل زمانی به وجود میاد که مقدار Learning Rate مدل خیلی بالا باشد یا مقدار Bias به میزان بسیار زیادی بزرگ و منفی باشد.**

2. به دلیل نزدیک بودن میانگین مقدار خروجی تابع Leaky RELU به صفر و این که اصطلاحا بالانس بودن نسبت به تابع RELU، سرعت یادگیری رو افزایش می دهد.

اما مشکل Vanishing در شبکه های عصبی با توجه به فرم تابع Leaky RELU، همچنان باقی می ماند.

در حقیقت تابع RELU به علت این که هم مقدار تابع و هم مشتقش بین 0 و 1 می باشند، عملا باعث جلوگیری از به وجود آمدن مشکل Vanishing می شود.