Propagarea erorilor Cum stratul `SoftMax` nu are parametri, în pasul *backward* trebuie calculat doar $\boldsymbol{\delta}_x$ pe baza $\boldsymbol{\delta}_y$. Formula 21 descrie acest calcul. Pentru o demonstrație matematică, citiți Secțiunea A.

$$\delta_{xk} = y_k \left(\delta_{y_k} - Z\right) \tag{21}$$

unde $Z = \sum_{k'} \delta_{y_{k'}} y_{k'}$.

4.1.4 Strat de linearizare

Pentru a putea trece de la straturi de convoluție la straturi complet conectate, *forma* activărilor trebuie schimbată din volum (tensor 3D) în vector. Într-un strat de tipul `Linearize` un volum $\mathbf{x} \in \mathbb{R}^{M \times H \times W}$ se transformă într-un vector $\mathbf{y} \in \mathbb{R}^{MHW}$. În pasul de propagare înapoi a gradienților, se petrece procesul invers: un vector $\boldsymbol{\delta}_y \in \mathbb{R}^{NHW}$ trebuie rearanjat într-un volum $\boldsymbol{\delta}_x \in \mathbb{R}^{N \times H \times W}$. Evident, acest strat nu are parametri și nu efectuează niciun calcul, doar schimbă aspectul unor tensori.

$$y_{m(HW)+iW+j} = x_{m,i,j} \quad 0 \leq m < M, 0 \leq i < H, 0 \leq j < W \tag{22}$$

$$\delta_{xm,i,j} = \delta_{y_{m(HW)+iW+j}}, \quad 0 \leq m < M, 0 \leq i < H, 0 \leq j < W \tag{23}$$

4.2 Rețele convoluționale

Rețelele convoluționale sunt rețele neurale dedicate imaginilor având o conectivitate specială ce urmărește obținerea echivarianței la translație. Rețelele convoluționale prelucrează volume de neuroni alternând straturi de convoluție, de redimensionare (*downsampling*) și `ReLU` (*rectified linear unit*). Spre final, rețelele convoluționale au unul sau două straturi complet conectate.

Consultați și pagina aceasta[4], una dintre cele mai bune prezentări ale rețelelor convoluționale.

4.2.1 Strat de convoluție

Straturile de convoluție `Conv` sunt specifice prelucrărilor de imagini, prelucrând neuroni dispuși în volume (tensori 3D). Volumele conțin hărți de caracteristici (*feature maps*) care reprezintă activări ale aceluiași detector în toate regiunile hărților de intrare. Acest lucru se întâmplă deoarece toți neuronii de pe o hartă au aceiași parametri.

Fiecare hartă descrie un filtru care se conectează pe o suprafață mică, dar pe toată adâncimea volumului de intrare. Un neuron se calculează printr-un produs scalar între parametrii lui (ai hărții căreia îi aparține) și neuronii din volumul de intrare de care este conectat. Ceilalți neuroni de pe aceeași hartă se calculează prin produse scalare între aceiași parametri și alte zone din volumul de intrare.

Un strat de convoluție este parametrizat de dimensiunea filtrului aplicat (***k**ernel size*) și de pasul (***s**tride*) dintre două aplicări ale filtrului.

Să presupunem că un astfel de strat primește la intrare M hărți de dimensiune $H \times W$. De exemplu, pentru o imagine RGB de dimensiune 32×32: $M = 3, H = 32, W = 32$. Mai departe să presupunem că fiecare hartă are un filtru de dimensiune $k = 3$ și îl aplică din 2 în 2 pixeli ($s = 2$). Dacă stratul calculează $N = 10$ hărți la ieșire, atunci dimensiunea volumului de ieșire va fi $N \times H' \times W'$:

[4] `http://cs231n.github.io/convolutional-networks/`

- adâncime: $N = 10$ hărți
- înălțime: $H' = (H - k)/s + 1$
- înălțime: $W' = (W - k)/s + 1$

Calculul realizat într-un strat de convoluție este cel din Formula 24. Fiecare hartă din volumul de ieșire este conectată la toate hărțile din volumul de intrare. De aceea, fiecare valoare din volumul de ieșire va depinde de $M \times k \times k$ valori din volumul de intrare.

$$y_{n,i,j} = \sum_{m=0}^{M-1} \sum_{p=0}^{k-1} \sum_{q=0}^{k-1} \theta_{n,m,p,q} \cdot x_{m,(i \cdot s)+p,(j \cdot s)+q} + b_n \tag{24}$$

Numărul de parametri ai unui strat de convoluție este $N \times (k \times k + 1)$.
Citiți și aici: `http://cs231n.github.io/convolutional-networks/#conv`.

Propagarea erorilor

$$\frac{\partial E}{\partial \theta_{n,m,p,q}} = \sum_{i,j} x_{n,m,i \cdot s+p,j \cdot s+q} \cdot \delta_{y_{n,i,j}} \tag{25}$$

$$\delta_{x_{m,i,j}} = \frac{\partial E}{\partial x_{m,i,j}} = \sum_{0 \leq n < N,p,q} \theta_{n,m,p,q} \cdot \delta_{y_{n,m,(i-p)/s,(j-q)/s}} \tag{26}$$

4.2.2 Max pooling

Calculul ieșirilor Un strat `MaxPooling` reduce dimensiunea unui volum de intrare într-unul cu același număr de hărți, dar de dimensiuni mai mici. Mai precis, fiecare hartă de intrare este împărțită în petice, iar fiecărui petic din hărțile de intrare îi corespunde o singură valoare la ieșire.

Stratul nu are parametri, dar are un hiper-parametru: latura unui petec (*stride*).

$$y_{m,i,j} = \max(x_{m,i \cdot stride,j \cdot stride}, \ldots x_{m,i \cdot (stride+1)-1,j \cdot (stride+1)-1}) \tag{27}$$

Propagarea erorilor În pasul de propagare înapoi a erorilor, gradientul *se transmite* intrării care a avut valoarea maximă într-un petice. Un gradient $\boldsymbol{\delta}_y$ de dimensiune $M \times (H/stride) \times (W/stride)$ este transformat într-un volum $\boldsymbol{\delta}_x$ de dimensiune $M \times H \times W$.

$$\delta_{x_{m,i,j}} = \begin{cases} \delta_{y_{m,\lfloor i/stride \rfloor,\lfloor j/stride \rfloor}} & \text{, daca } \delta_{x_{m,i,j}} == y_{m,\lfloor i/stride \rfloor,\lfloor j/stride \rfloor} \\ 0 & \text{, altfel} \end{cases} \tag{28}$$

4.2.3 ReLU

Calculul ieșirilor Un strat de tip `ReLU` (*rectified linear unit*) primește un tensor $\mathbf{x}$ și produce la ieșire un tensor $\mathbf{y}$ cu aceleași dimensiuni. O valoare de ieșire y_i se calculează conform formulei 29.

$$y_i = relu(x_i) = \max(x_i, 0) \tag{29}$$

Straturile de tip `ReLU` nu au parametri.