

Härledning av konvolutionella neurala nätverk för ansiktsigenkänning och sifferavläsning

Nikita Zozoulenko

25 januari 2018

Abstract

Convolutional Neural Networks are a biologically inspired machine learning model that was popularized 2012 when it won the annual ImageNet Large-Scale Visual Recognition Challenge, beating all current machine learning models. Today they have achieved state of the art results in areas such as self-driving cars, image classification, object localization, automatic image annotation, semantic segmentation and natural language processing. In this paper the mathematical model behind the standard Artificial Neural Network and the Convolutional Neural Network is derived. A Convolutional Neural Network is then adapted to read handwritten digits and to make a state of the art face detector, successfully detecting over 200 faces in a crowded scene.

Innehåll

1	Inledning	4
1.1	Bakgrund	4
1.2	Syfte	4
1.3	Frågeställning	4
2	Metod	5
3	Bakgrund	6
3.1	Tensorer, indexering och notation	6
3.2	Feed-forward neurala nätverk	6
3.2.1	Framåtpropagering	6
3.2.2	Kostnadsfunktionen	8
3.2.3	Gradient Descent	9
3.2.4	Bakåtpropagering	9
3.3	Träning av neurala nätverk	11
4	Resultat	13
4.1	Konvolutionella neurala nätverk	13
4.1.1	Konvolutionslagret framåtpropagering	15
4.1.2	Konvolutionslagret bakåtpropagering	16
4.1.3	Aktiveringsfunktionslager framåtpropagering	18
4.1.4	Aktiveringsfunktionslager bakåtpropagering	18
4.1.5	Maxpoollagret framåtpropagation	19
4.1.6	Maxpoollagret bakåtpropagering	20
4.1.7	Batch Normalization framåtpropagering	20
4.1.8	Batch Normalization bakåtpropagering	22
4.1.9	Softmax framåtpropagering	25
4.1.10	Softmax bakåtpropagering	25
4.2	Praktiska Tillämpningar	26
4.2.1	Klassificering av handskrivna siffror	26
4.2.2	Objektdetektering i bilder	29
4.2.3	Ansiktsigenkänning	29
5	Diskussion	30

1 Inledning

1.1 Bakgrund

Artificiella neurala nätverk var först introducerade på 1940-talet och simulerades med hjälp av strömkretsar. Modellen är inspirerad från hur människors och djurs hjärna är uppbyggda av ett flertal neuroner som är kopplade till varandra och överför en nervsignal. Yann LeCun uppfann det Konvolutionella Neurala Nätverket 1998 och den var designad till att efterlikna hur ett djurs prefrontalkortex fungerar. Trots att modellen hade funnits i flera år hade den inte sett stor användning innan år 2012 när ett konvolutionellt neuralt nätverk användes för att vinna den årliga ImageNet objektklassifikationstävlingen. Under de senaste 5 åren har modellen använts för att uppnå de bästa möjliga resultaten inom bland annat självkörande bilar, objektdetektering i bilder, meningsöversättning mellan språk och automatiskt skapande av undertexter från ljud.

1.2 Syfte

Syftet med arbetet är att härleda och redogöra för den underliggande matematiken bakom den matematiska modellen av konvolutionella neurala nätverk, samt att visa exempel på praktiska tillämpningar av modellen.

1.3 Frågeställning

Vad är ett konvolutionellt neuralt nätverk? Hur härleds framåt- och bakåtpropagering i konvolutionella neurala nätverk? Hur kan modellen tillämpas för att implementera sifferavläsning, ansiktsigenkänning och objektdetektering i bilder?

2 Metod

Metod ej klar

Majoriteten av tiden gick åt till se på videoföreläsningar av kursen "CS231n: Convolutional Neural Networks for Visual Recognition" från Stanford University för att få en grundläggande bakgrund och intuition om artificiella neurala nätverk. Jag implementerade sedan ett klassiskt artificiellt neuralt nätverk i C++ och python, och ett konvolutionellt neuralt nätverk i python. De partiella derivatorna som måste beräknas i modellen jämfördes med en analyti

3 Bakgrund

3.1 Tensorer, indexering och notation

Tensorer är generaliseringen av vektor- och matrisbegreppen. En tensor med ordning 1 är en vektor $x \in \mathbb{R}^N$ och är en radvektor med N element. Det kan dessutom tolkas som en endimensionell array. Matriser M är tensorer med ordning 2 sådana att $M \in \mathbb{R}^{R \times N}$ och kan ses som en vektor av R vektorer med N element eller en tvådimensionell array med $R \times N$ element. En tensor med ordning n indexeras med en n -tupel. Exempelvis indexeras en tensor $X \in \mathbb{R}^{R \times C \times H \times W}$ med fyr-tupeln (r, c, h, w) där $0 \leq r < R$, $0 \leq c < C$, $0 \leq h < H$ och $0 \leq w < W$. [1]

3.2 Feed-forward neurala nätverk

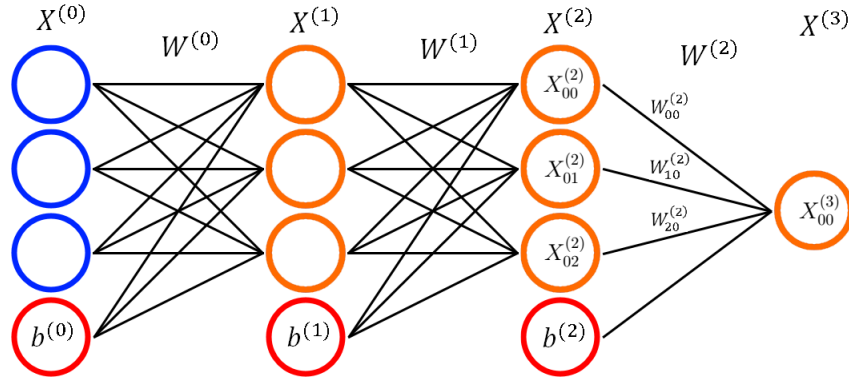
Ett artificiellt neuralt nätverk består av ett antal lager neuroner. De är uppbyggda rekursivt så att resultatet av ett lager är inmatningsdatan till nästintilliggande lager. Inmatningsdatan för modellen matas in i det första lagret sådant att ett neuron är ett värde. Nervsignalen propageras framåt tills den når det sista lagret. Hur nervceller från ett lager är kopplade till det föregående lagret varierar med vilket typ av neuralt nätverk man använder sig av. [1]

Den mest grundläggande modellen har flera namn, bland annat: *Multilayer Perceptrons Fully Connected Cascade (FCC)*, *Feed-forward Neural Network* och *Densely Connected (Dense)*. Den består av ett flertal lager av neuroner. Varje neuron i ett lager är kopplade till alla neuroner i nästintillföljande lager. Nervsignalen framåtpropageras beroende på hur stark kopplingen mellan två neuroner är. Ett neurons värde kallas för dess aktivering. [1]

3.2.1 Framåtprogagering

Forward propagation eller framåtpropagation är processen att från sina inputneuroner propagera nervsignalen framåt i nätverket tills man når det sista lagret. [1]
[6]

Nätverket kan framställas genom att representera neuronerna och deras kopplingar med hjälp av matriser. Flera tränings exempel bearbetas samtidigt i en så kallad mini-hop med storlek R . Låt $X_{ri}^{(l)}$ benämna neuron nummer i i lager l i tränings exempel r . $W_{ba}^{(l)}$ blir vikten eller styrkan på kopplingen mellan



Figur 1: Ett exempel på ett enkelt feed-forward neuralt nätverk. Inputneuronerna är blåmarkerade medan resterande neuroner är orangea. Röda neuroner är så kallade *bias-neuroner* som är konstanta oberoende på inmatningsdatan. Svarta linjer symboliserar vikterna och styrkan mellan två neuroner.

neuron $X_{ra}^{(l)}$ och $X_{rb}^{(l+1)}$. Låt $b^{(l)}$ vara ett konstant neuron kallat *bias* som är kopplad till alla neuron i lager $l + 1$. [1] [6]

Värdet för ett godtyckligt neuron kallas för den neuronens aktivering. För att beräkna aktiveringen av ett neuron i lager $l + 1$ multipliceras varje neurons värde i lager l med deras korresponderande vikt i viktmatrisen $W^{(l)}$. Resultaten och lagrets konstanta neuron $b^{(l)}$ summeras och summan matas in i en aktiveringsfunktion $f(x)$. Funktionsvärdet blir aktiveringen för det nya neuronet i lager $l + 1$. Exempelvis kan signalöverföringen mellan lager 2 och 3 för ett godtyckligt exempel r i mini-hopen i figur 1 beskrivas matematiskt genom: [1] [6]

$$Z_{r0}^{(3)} = X_{r0}^{(2)} W_{r0}^{(2)} + X_{r1}^{(2)} W_{10}^{(2)} + X_{r2}^{(2)} W_{20}^{(2)} + b^{(2)} \quad (1)$$

$$X_{r0}^{(3)} = f(Z_{r0}^{(3)}) \quad (2)$$

Där $X^{(3)} \in \mathbb{R}^{1 \times 1}$, $X^{(2)} \in \mathbb{R}^{1 \times 3}$, $W^{(2)} \in \mathbb{R}^{3 \times 1}$ och $b^{(2)} \in \mathbb{R}^1$.

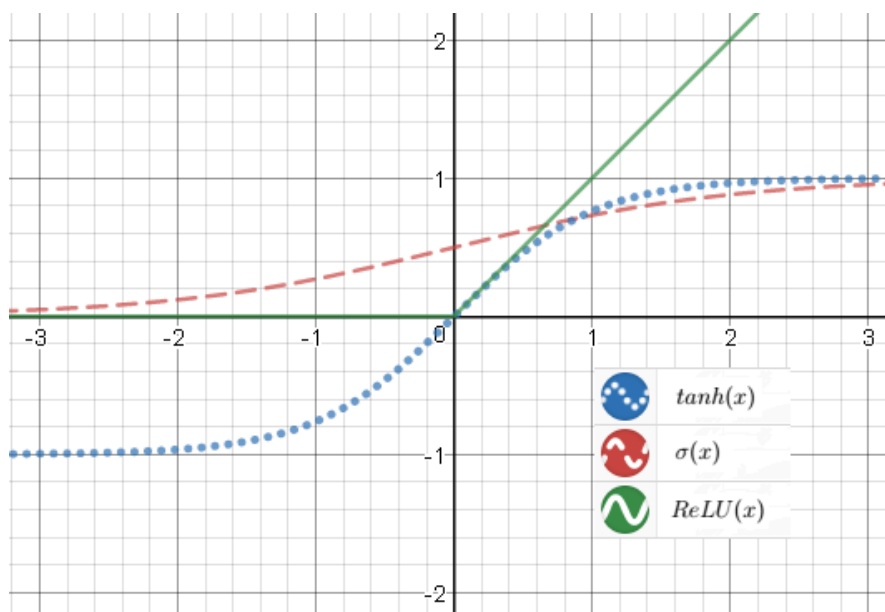
Neuronmatriserna och viktmatriserna är definierade på ett sådant sätt att framåtpropagationen mellan lager l och $l+1$ kan beräknas med en matrismultiplikation och kan därför beräknas rekursivt från inputneuronerna: [1] [6]

$$Z_{r,i}^{(l+1)} = [X^{(l)} W^{(l)}]_{r,i} + b^l \quad (3)$$

$$X_{r,i}^{(l+1)} = f(Z_{r,i}^{(l+1)}) \quad (4)$$

Vanliga aktiveringsfunktioner för neurala nätverk är *Rectified Linear Units (ReLU)*, *sigmoid (σ)* och *tangens hyperbolicus (\tanh)*. Funktionerna måste vara deriverbara för att nätverket ska kunna tränas genom processen som

kallas för bakåtpagering. Genom att använda olinjära funktioner kan nätverket lära sig olinjära samband. Definitionerna av funktionerna ges av: [1]



Figur 2: Grafen av aktiveringsfunktionerna $ReLU$, σ och \tanh .

$$ReLU(x) = \begin{cases} 0 & \text{om } x < 0 \\ x & \text{om } x \geq 0 \end{cases} \quad (5)$$

$$\sigma(x) = \frac{1}{1 + e^{-x}} \quad (6)$$

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad (7)$$

3.2.2 Kostnadsfunktionen

Givet ett inmatningsvärde X skapas ett närmevärde \hat{y} som ska vara så nära det sanna värdet för y som möjligt. Initialt kommer $W^{(l)}$ vara slumpade för varje lager l och nätverkets prognos kommer inte efterlikna det sökta värdet. Med hjälp av *gradient descent* kan man iterativt träna modellen så det slutgiltiga värdet kommer så nära y som möjligt. Detta görs genom att definiera en multivariat kostnadsfunktion $L(W, b; X, y)$ av alla nätverkets parametrar med avseende på ett träningsexempel (X, y) . Funktionen är ett mått på prognosen \hat{y} kvalitet. Man definierar L på ett sådant sätt att ju

mindre värdemängd av L , desto högre kvalitet består \hat{y} av. Ett sätt att definiera L är exempelvis med en så kallad *L2 kostnadsfunktion*: [1] [6]

$$\begin{aligned} L(\theta) &= \|\hat{y} - y\|^2 \\ &= \|f(f(f(XW^{(0)} + b^{(0)})W^{(1)} + b^{(1)})W^{(2)} + b^{(2)}) - y\|^2 \end{aligned} \quad (8)$$

Träningen av det neurala nätverket är processen där man hittar kostnadsfunktionens minimum med hjälp av *gradient descent*.

3.2.3 Gradient Descent

Gradienten $\nabla L(\theta)$ är en vektor av partiella derivator med avseende på funktionen L variabler $W^{(0)}, W^{(1)}, \dots, W^{(l)}, b^{(0)}, b^{(1)}, \dots, b^{(l)}$ som definieras genom: [7] [3]

$$\nabla L(\theta) : \mathbb{R}^n \rightarrow \mathbb{R}^n \quad (9)$$

$$\nabla L(\theta) = \left(\frac{\partial L(\theta)}{\partial \theta^{(0)}} \quad \frac{\partial L(\theta)}{\partial \theta^{(1)}} \quad \dots \quad \frac{\partial L(\theta)}{\partial \theta^{(n-1)}} \right) \quad (10)$$

Gradienten $\nabla L(\theta)$ visar riktningen vari värdemängdsökningen är som störst. Genom att ändra vikterna θ värde proportionellt med avseende på den negativa gradienten $-\nabla L(\theta)$ kan man iterativt modifiera θ tills man når funktionens minimum. Den mest grundläggande algoritmen för *gradient descent* kallas för *Stochastic Gradient Descent (SGD)* och använder hyperparametern α för att beteckna träningshastigheten: [7] [3] [6]

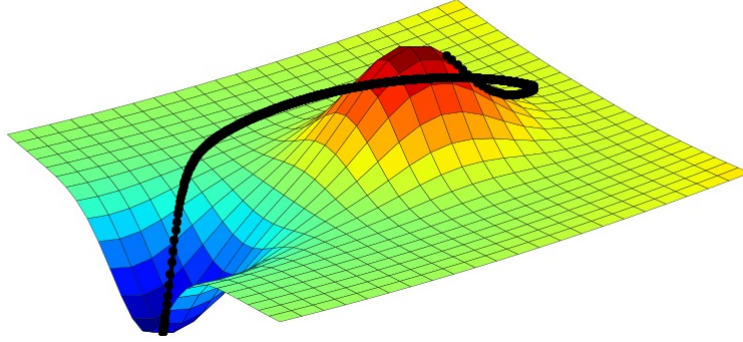
$$\frac{\partial L(\theta)}{\partial \theta^{(l)}} = \nabla_{\theta^{(l)}} L(\theta) \quad (11)$$

$$\theta^{(l)} \rightarrow \theta^{(l)} - \alpha \frac{\partial L(\theta)}{\partial \theta^{(l)}} \quad (12)$$

Nätverket tränas genom att varje träningsexempel (X, y) framåtpropageras och insätts i kostnadsfunktionens L . Genom processen som kallas för bakåtpropagering beräknas alla partiella derivator med avseende på en viss vikt och används för att uppdatera vikterna tills nätverkets prognos liknar de sanna värdena. [7] [3] [6]

3.2.4 Bakåtpropagering

De partiella derivatorna kan approximeras med hjälp av framåt, bakåt eller central differenskvot för partiella derivator: [6] [7]



Figur 3: En illustration av gradient descent på en funktion med två variabler.[18]

$$\frac{\partial L(\theta)}{\partial \theta^{(i)}} = \frac{L(\theta^{(0)}, \dots, \theta^{(i)} + h, \dots, \theta^{(n-1)}) - L(\theta)}{h} \quad (13)$$

Detta skulle inte skapa några problem för det lilla neurala nätverket i figur 1 med 24 parametrar, men i själva verket har djupa neurala nätverk miljontals av parametrar som behöver en enorm datorkraft. Istället tillämpas regler för differentialkalkyl för att effektivt beräkna de partiella derivatorna. För ett godtyckligt lager l används kedjeregeln för att bryta upp derivatan av L med avseende på en vikt i tre bråk. På grund av att alla träningsexempel i mini-hopen använder vikten i fråga summeras alla exempel i mini-hopen R . För derivatan av de konstanta neuronerna summeras alla partiella derivator för det lagret. För att sambandet mellan två neuroner och en godtycklig vikt är linjär kan det enkelt deriveras. [1]

Låt

$$\frac{\partial L(\theta)}{\partial X^{(l)}} = \delta^{(l)} \quad (14)$$

Derivatan av L med avseende på ett lagrets parameter beräknas genom: [1]

$$\begin{aligned} \frac{\partial L(\theta)}{\partial W_{j,i}^{(l)}} &= \sum_{r=0}^{R-1} \frac{\partial L(\theta)}{\partial X_{r,i}^{(l+1)}} \frac{\partial X_{r,i}^{(l+1)}}{\partial Z_{r,i}^{(l+1)}} \frac{\partial Z_{r,i}^{(l+1)}}{\partial W_{i,j}^{(l)}} \\ &= \sum_{r=0}^{R-1} \delta_{r,i}^{(l+1)} f'(Z_{r,i}^{(l+1)}) X_{r,j}^{(l)} \end{aligned} \quad (15)$$

$$\begin{aligned}
\frac{\partial L(\theta)}{\partial b^{(l)}} &= \sum_{r=0}^{R-1} \frac{\partial L(\theta)}{\partial X_{r,i}^{(l+1)}} \frac{\partial X_{r,i}^{(l+1)}}{\partial Z_{r,i}^{(l+1)}} \frac{\partial Z_{r,i}^{(l+1)}}{\partial b_{i,j}^{(l)}} \\
&= \sum_{r=0}^{R-1} \delta_{r,i}^{(l+1)} f'(Z_{r,i}^{(l+1)})
\end{aligned} \tag{16}$$

$\delta^{(l)}$, också kallat *delta-felet*, är enligt dess definition derivatan av L med avseende på ett visst neuron. Det kan tolkas som hur känslig värdemängdsförändringen för kostnadsfunktionen är för ett visst neuron vid lager l . En vikt kopplat till ett neuron med högt delta-fel är mer ansvarig för skiftning i kostnaden än ett neuron med lågt delta-fel. Genom att propagera nervsignalen baklänges i nätverket med kostnadsfunktionens värde som inputvärde kan man beräkna alla neuroner $X^{(l)}$ påverkan på L vid ett godtyckligt lager l . [1]

Bakåtpropageringen av delta-felen uttrycks rekursivt från det sista lagret \hat{y} . Delta-felet vid det sista lagret l_{sista} beror på kostnadsfunktionen. För en L2 kostnadsfunktion definieras delta-felet som: [1]

$$\begin{aligned}
\delta^{(l_{sista})} &= \frac{\partial L(\theta)}{\partial \hat{y}} \\
&= 2||\hat{y} - y||
\end{aligned} \tag{17}$$

Likt ekvation (15) summeras alla partiella derivator av neuronerna vid användningen av kedjeregeln: [1]

$$\begin{aligned}
\delta_{r,i}^{(l)} &= \frac{\partial L(\theta)}{\partial X_{r,i}^{(l)}} \\
&= \sum_{j=0}^{N'} \frac{\partial L(\theta)}{\partial X_{r,j}^{(l+1)}} \frac{\partial X_{r,j}^{(l+1)}}{\partial Z_{r,j}^{(l+1)}} \frac{\partial Z_{r,j}^{(l+1)}}{\partial X_{r,i}^{(l)}} \\
&= \sum_{j=0}^{N'} \delta_{r,j}^{(l+1)} f'(Z_{r,i}^{(l+1)}) W_{j,i}^{(l)}
\end{aligned} \tag{18}$$

Bakåtpropageringen sker lager till lager med start i outputlagret. Vid varje lager måste delta-felet $\delta^{(l)}$ bakåtpropageras och derivatan av det lagrets parametrar $W^{(l)}$ och $b^{(l)}$ beräknas. [1]

3.3 Träning av neurala nätverk

Neurala nätverk tränas genom att iterativt uppdatera nätverkets vikter med hjälp av *gradient descent*. Den hela träningsdatabasen delas in i mini-hopar

och väljs slumpvist ut. En iteration av SGD appliceras genom att framåtpropagera och sedan bakåtpropageras träningssexemplet för att beräkna gradienten av kostnadsfunktionen med avseende på alla vikter i nätverket. Denna process upprepas tills nätverkets vikter konvergerar och nätverkets prognos efterliknar verkligheten. [1]

Två implementationer av ett feed-forward neuralt nätverk kan hittas på github i python och C++: <https://github.com/nikitazozoulenko>

4 Resultat

4.1 Konvolutionella neurala nätverk

När människor vill identifiera någonting i en bild så letar vi efter vissa karakteristiska drag objektet har. En hund består exempelvis av en kropp, ett huvud och fyra ben. Kroppsdelarna består sedan själva av grundläggande geometriska former som i sig självt är kombinationer av kanter och linjer. Dessutom har hundar en viss textur, det som vi kännetecknar som något pälsliknande. Dessa karakteristiska drag är lokala inom bilden och kan extraheras av att endast se på en liten del av bilden i taget. Detta är principen bakom *Konvolutionella Neurala Nätverk (CNN)*: Genom så kallade *konvolutioner* kunna extrahera dessa karakteristiska drag. Nätverket lär sig ett antal små filter som den applicerar på en delmängd av bilden genom att filtret sammanrullar över hela bilden (se figur 4). Värdet av filtret över en delmängd av bilden blir aktiveringen av ett neuron i nästa lager. [1]



Figur 4: Resultatet av att ett filter för vertikal respektive horisontell kantdetektering har sammanrullat över en bild av en katt.

Till skillnad från *FCC* är neuronerna i ett *CNN* bara kopplade till närliggande neuronerna i det föregående lagret. På detta sätt kan nätverket lära sig fler hög-nivåspecialartiklar ju djupare i nätverket signalen går. Exempelvis kan det hända att det första lagret lär sig att identifiera kanter och linjer medan de senare lagren lagren lär sig att känna igen olika geometriska former och till sist känna igen ansikten eller object i det sista lagret. [1]

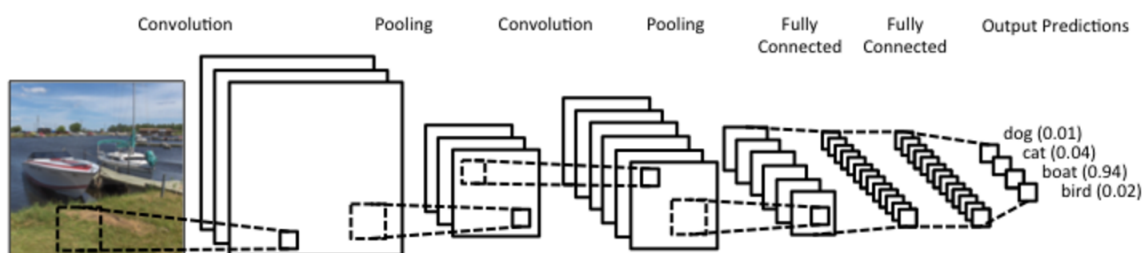
Modellen, precis som ett *feed-forward nätverket*, består av ett flertal lager neuroner sådant att resultatet av ett lager matas in till nästkommande lager. Det sista lagret benämns med $X^{(l_{sista})}$ eller \hat{y} . Vid varje lager finns dessutom vikter $W^{(l)}$ som beror på vad för slags lager det är. $W^{(l)}$ kan vara tom med inga vikter när lager inte bidrar till någon inlärning. Ekvation (19) illustrerar strukturen av modellen.[1] [3]

$$X^{(0)} \xrightarrow{W^{(l)}} X^{(1)} \xrightarrow{W^{(l)}} \dots \xrightarrow{W^{(l_{sista}-2)}} X^{(l_{sista}-1)} \xrightarrow{W^{(l_{sista}-1)}} X^{(l_{sista})} = \hat{y} \quad (19)$$

Samtliga olika typer av lager i en konvolutionellt neuralt nätverk har två olika faser: framåtpropagering och bakåtpropagering. Vid framåtpropagering används aktiveringen av det föregående lagret för att beräkna aktiveringen vid nästintilliggande lager. Vid bakåtpropagering måste det rekursiva delta-felet propageras baklänges i nätverket med hjälp av kedjeregeln för att sedan beräkna den partilla derivatan av lagrets vikter med avseende på kostnadsfunktionen. [1]

För ett feed-forward neuralt nätverk används matriser för att representera neuronerna. I ett *CNN* är en tensor $X^{(l)} \in \mathbb{R}^{R \times C \times H \times W}$ med ordning 4 aktiveringen vid lager l och indexeras med fyr-tupeln (r, c, h, w) där $0 \leq r < R$, $0 \leq c < C$, $0 \leq h < H$ och $0 \leq w < W$. Aktiveringarna vid nästintilliggande lager $l + 1$ betecknas med $X^{(l+1)} \in \mathbb{R}^{R' \times C' \times H' \times W'}$ och indexeras med fyr-tupeln (r', c', h', w') där $0 \leq r' < R'$, $0 \leq c' < C'$, $0 \leq h' < H'$ och $0 \leq w' < W'$. [1] [3]

Aktiveringen brukar illustreras som en tredimensionell volym där W , H och C är bredden, höjden respektive djupet. En $H \times W$ skiva av volymen kallas för en *feature map* eller en *kanal*. Antalet kanaler betecknas med C . R står för hopstorlek då man bearbetar R exempel i taget i en så kallad mini-hop. [1] [3]



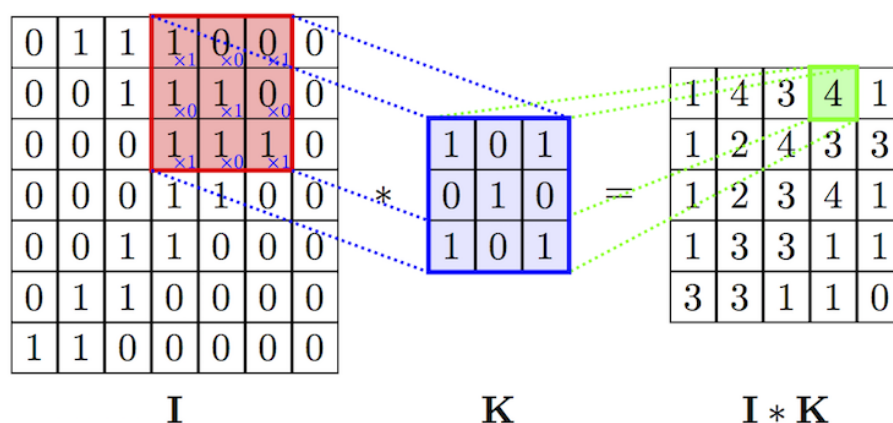
Figur 5: En illustration av ett konvolutionellt neuralt nätverk. Varje skiva är en egen *feature map*. [19]

På grund av att en konvolution är en lokal operator används CNN:s för data som innehåller lokalt sammanhängande samband, exempelvis bilder eller ljud. Om det är en bild som bearbetas har det första lagrets aktivering $C = 3$ kanaler, en för varje RGB-kanal, och en bredd och höjd lika med bildens bredd och höjd i pixlar. [1] [3]

4.1.1 Konvolutionslagret framåtpropagering

Ett konvolutionslager består av ett antal vikter kallade *kärnor* (*kernels*) eller *masker* (*masks*), representerade av en tensor med ordning fyra, $W^{(l)} \in \mathbb{R}^{C' \times C \times K_h \times K_w}$ för lager l . [1] [3]

När masken är över en godtycklig del av volymen multipliceras varje värde i delmängden av $W^{(l)}$ elementvis med respektive värde i masken vid samma position och summeras (se figur 6). Summan blir aktiveringen av ett neuron i nästa lager. Konvolutionsoperatoren betecknas med $*$. [1] [3]



Figur 6: En kärna med storlek 3×3 sammanrullar över ett område med dimensioner 6×6 och bildar en aktivering med dimensionerna 4×4 . [20]

En feature map i lager $l + 1$ är resultatet av att en kärna med dimensioner $1 \times \times C \times K_h \times K_w$ har sammanrullat över aktiveringen av det föregående lagret. C' är antalet kärnor och blir dessutom antalet feature maps nästa lager har. [1] [3]

Kärnorna har två ytterligare egenskaper: ett kliv s och så kallad *zero-padding* p . s är hur stort kliv man tar efter varje gång filtret blir applicerat på tensorn. Man ökar tensors höjd och bredd med $2p$ genom att fylla på med nollor vid tensors ändor (se figur 7). På grund av att aktiveringens höjd och bredd avtar ju djupare i nätverket de befinner sig på används zero-padding för att kontrollera storleken av tensorn. [1] [3] [4]

0	0	0	0	0
0	1	2	3	0
0	2	3	1	0
0	4	6	2	0
0	0	0	0	0

Figur 7: Ett område med dimensioner 3×3 zero-paddas med $p = 1$ och resulterande område får dimensioner 5×5 .

Låt $W^{(l)} \in \mathbb{R}^{C' \times C \times K_h \times K_W}$, $X^{(l)} \in \mathbb{R}^{R \times C \times (H+2p) \times (W+2p)}$ och $X^{(l+1)} \in \mathbb{R}^{R \times C' \times H' \times W'}$. Dimensionerna vid lager $l + 1$ beskrivs av: [1] [3] [4]

$$W' = \frac{W - K_W + 2p}{s} + 1 \quad (20)$$

$$H' = \frac{H - K_H + 2p}{s} + 1 \quad (21)$$

Då beskrivs en konvolution algebraiskt genom: [1] [3]

$$w = sw' \quad (22)$$

$$h = sh' \quad (23)$$

$$\begin{aligned} [X^{(l+1)}]_{r,c',h',w'} &= X_{r,c',h',w'}^{(l)} * W_{c'}^{(l)} \\ &= \sum_{c=0}^{C-1} \sum_{j=0}^{K_H-1} \sum_{i=0}^{K_W-1} X_{r,c,h'+j,w'+i}^{(l)} W_{c',c,j,i}^{(l)} \end{aligned} \quad (24)$$

Index på termen som ska sammanrullas i konvolutionen symboliserar vilka dimensioner som ska summeras. Exempelvis visar $W_{c'}^{(l)}$ att dimensionerna C , H och W (alla kanaler) ska summeras medan $W_{c',c}^{(l)}$ visar att endast H och W (en kanal) ska summeras.

I praktiken brukar konvolutioner implementeras med hjälp av funktionerna *row2im* och *im2row* vilka möjliggör att en konvolution att beräknas med en matrismultiplikation. [1] [3] [4]

4.1.2 Konvolutionslagret bakåtpropagering

Vid varje lager l bakåtpropageras det rekursiva delta-felet δ och den partiella derivatan av lagerts vikter med avseende på kostnadsfunktionen L beräknas.

Bakåtpropageringen av det rekursiva delta-felet $\frac{\partial L(W)}{\partial X_{r,c',h',w'}^{(l+1)}}$ räkas ut med hjälp av kedjeregeln. Derivatans kan inte endast delas upp i $\frac{\partial L(W)}{\partial X_{r,c',h',w'}^{(l+1)}}$ och $\frac{\partial X_{r,c',h',w'}^{(l+1)}}{\partial X_{r,c,h,w}^{(l)}}$, utan alla derivator måste summeras på grund av att det är mer än ett neuron som är ansvarig för framåtpropageringen likt ekvationer (15), (16) och (18). $X_{r,c',h',w'}^{(l+1)}$ byts sedan ut mot dess definition enligt ekvation (24). [3] [11] [13] [15]

$$\begin{aligned}
\delta_{r,c,h,w}^{(l)} &= \frac{\partial L(W)}{\partial X_{r,c,h,w}^{(l)}} \\
&= \sum_{c'=0}^{C'-1} \sum_{h'=0}^{H'-1} \sum_{w'=0}^{W'-1} \frac{\partial L(W)}{\partial X_{r,c',h',w'}^{(l+1)}} \frac{\partial X_{r,c',h',w'}^{(l+1)}}{\partial X_{r,c,h,w}^{(l)}} \\
&= \sum_{c'=0}^{C'-1} \sum_{h'=0}^{H'-1} \sum_{w'=0}^{W'-1} \delta_{r,c',h',w'}^{(l+1)} \frac{\partial \sum_{c=0}^{C-1} \sum_{j=0}^{K_H-1} \sum_{i=0}^{K_W-1} X_{r,c,h'+j,w'+i}^{(l)} W_{c',c,j,i}^{(l+1)}}{\partial X_{r,c,h,w}^{(l)}}
\end{aligned} \tag{25}$$

Varje produkt i den innersta summan kommer att vara lika med noll förutom om $X_{r,c,h'+j,w'+i}^{(l)} = X_{r,c,h,w}^{(l)}$. Förljaktligen insätter man $h' + j = h$ och $h' + j = h$. Summorna och derivatan förkortas: [11] [13] [15]

$$\begin{aligned}
&\sum_{c'}^{C'-1} \sum_{h'=0}^{H'-1} \sum_{w'=0}^{W'-1} \delta_{r,c',h',w'}^{(l+1)} \frac{\partial \sum_{c=0}^{C-1} \sum_{j=0}^{K_H-1} \sum_{i=0}^{K_W-1} X_{r,c,h'+j,w'+i}^{(l)} W_{c',c,j,i}^{(l+1)}}{\partial X_{r,c,h,w}^{(l)}} \\
&= \sum_{c'=0}^{C'-1} \sum_{h'=0}^{H'-1} \sum_{w'=0}^{W'-1} \delta_{r,c',h',w'}^{(l+1)} W_{c',c,j,i}^{(l+1)} \\
&= \sum_{c'=0}^{C'-1} \sum_{h'=0}^{H'-1} \sum_{w'=0}^{W'-1} W_{c',c,(h-h'),(w-w')}^{(l+1)} \delta_{r,c',h',w'}^{(l+1)}
\end{aligned} \tag{26}$$

Vilket man kan se är en summa av konvolutioner där en viss *feature map* av delta-felet sammanrullar över alla kärnor på en viss *feature map* med vikter som är roterade 180°. För att en konvolution ska kunna ske måste den roterade vikten zero-paddas på grund av att det glidande fönstret måste vara som mest lika stor som tensorn den sammanrullar över. Låt rotationen betecknas med funktionen $rot()$. [11] [13] [15]

$$\delta_{r,c,h,w}^{(l)} = \sum_{c'=0}^{C'-1} rot(W_{c',c,h,w}^{(l+1)}) * \delta_{r,c'}^{(l+1)} \tag{27}$$

En sundshetskontroll visar att detta är intuitivt då alla *feature maps* i $X^{(l)}$ används för att skapa en enstaka *feature map* i $X^{(l+1)}$. Det är därför man summerar över alla kärnor och endast konvolverar i en *feature map* i taget och summerar alltihop. [1]

Den partiella derivatan av kostandsfunktionen med avseende på vikterna hittas på ett liknande sätt. Här summeras dessutom R -dimensionen på grund av att alla exempel i mini-hopen har en påverkan på gradienten. Ytterligare divideras gradienten med R för att hitta medelvärdet av alla gradienter av alla exempel i mini-hopen: [1] [11] [13] [15]

$$\begin{aligned}
\frac{\partial L(W)}{\partial W_{c',c,h,w}^{(l)}} &= \frac{1}{R} \sum_{r=0}^{R-1} \sum_{c'=0}^{C'-1} \sum_{h'=0}^{H'-1} \sum_{w'=0}^{W'-1} \frac{\partial L(W)}{\partial X_{r,c',h',w'}^{(l+1)}} \frac{\partial X_{r,c',h',w'}^{(l+1)}}{\partial W_{r,c,h,w}^{(l)}} \\
&= \frac{1}{R} \sum_{r=0}^{R-1} \sum_{c'=0}^{C'-1} \sum_{h'=0}^{H'-1} \sum_{w'=0}^{W'-1} \delta_{r,c',h',w'}^{(l+1)} \frac{\partial \sum_{c=0}^{C-1} \sum_{j=0}^{K_H-1} \sum_{i=0}^{K_W-1} X_{r,c,h'+j,w'+i}^{(l)} W_{c',c,j,i}^{(l)}}{\partial W_{c',c,h,w}^{(l)}} \\
&= \frac{1}{R} \sum_{r=0}^{R-1} \sum_{c'=0}^{C'-1} \sum_{h'=0}^{H'-1} \sum_{w'=0}^{W'-1} X_{r,c,h'+h,w'+w}^{(l)} \delta_{r,c',h',w'}^{(l+1)} \\
&= \frac{1}{R} \sum_{r=0}^{R-1} \sum_{c'=0}^{C'-1} X_{r,c,h,w}^{(l)} * \delta_{r,c'}^{(l+1)}
\end{aligned} \tag{28}$$

4.1.3 Aktiveringsfunktionslager framåtpropagering

Aktiveringsfunktionen f appliceras elementvis på alla neuroner i $X^{(l)}$. Följaktligen har $X^{(l)}$ och $X^{(l+1)}$ samma dimensioner. Låt aktiveringsfunktionen betecknas med f . Nervsignalen framåtpropageras genom: [3]

$$X_{r,c,h,w}^{(l+1)} = f(X_{r,c,h,w}^{(l)}) \tag{29}$$

Aktiveringsfunktioner ökar nätverks precision och får dem att konvergera snabbare, vilket leder till att mindre datakraft krävs för att träna nätverket. [1]

4.1.4 Aktiveringsfunktionslager bakåtpropagering

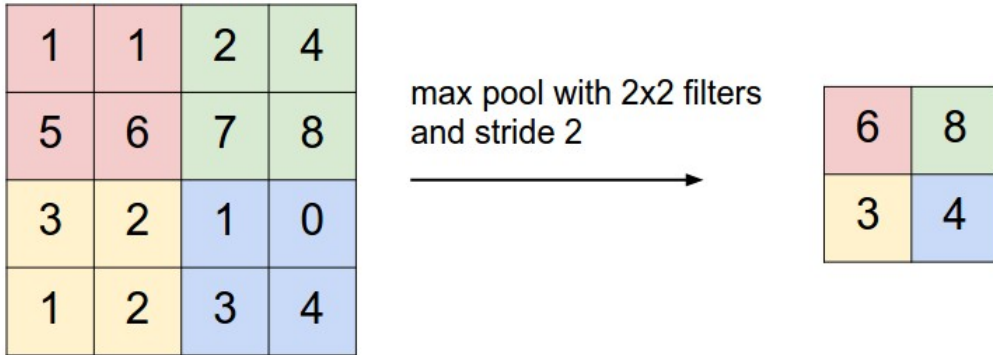
Aktiveringsfunktioner har inga parametrar som ska optimeras och sålades är $W^{(l)}$ och $\frac{\partial L(W)}{\partial W^{(l)}}$ tomma. Bakåtpropageringen av det rekursiva delta-felet

härleds med hjälp av kedjeregeln: [1] [3]

$$\begin{aligned}
\delta_{r,c,h,w}^{(l)} &= \frac{\partial L(W)}{\partial X_{r,c,h,w}^{(l)}} \\
&= \frac{\partial L(W)}{\partial X_{r,c,h,w}^{(l+1)}} \frac{\partial X_{r,c,h,w}^{(l+1)}}{\partial X_{r,c,h,w}^{(l)}} \\
&= \delta_{r,c,h,w}^{(l+1)} f'(X_{r,c,h,w}^{(l)})
\end{aligned} \tag{30}$$

4.1.5 Maxpoolaget framåtpropagation

Här är igen inputneuronerna representerade av $X^{(l)} \in \mathbb{R}^{R \times C \times H \times W}$ och skapar output $X^{(l+1)} \in \mathbb{R}^{R \times C' \times H' \times W'}$. Lagret saknar vikter men har däremot hyperparametrarna k (kärnstorlek) och s (stride eller kliv). *Maxpooling* delar in varje *feature map* i $X^{(l)}$ i ett antal sektioner med dimensioner $k \times k$ genom att ett glidande fönster med samma dimensioner sammanrullar över alla lagrets *feature maps* (se figur 8). Aktiveringen vid ett neuron i lager $l+1$ blir lika med det största värdet i korresponderande $k \times k$ sektion. [1] [3] [4]



Figur 8: Maxpooling med $k = 2$ och $s = 2$ av ett område med dimensioner 4×4 där resultatet bildar ett område med dimensionerna 2×2 .

Liknande konvolutionslagret utan zero-padding blir det nästintillkommande lagrets dimensioner: [1] [3] [4]

$$W' = \frac{W - k}{s} + 1 \tag{31}$$

$$H' = \frac{H - k}{s} + 1 \tag{32}$$

$$C' = C \tag{33}$$

Antal kanaler förblir konstant. [1] [3]

Matematiskt beskrivs maxpoollagret genom: [1] [3]

$$X_{r,c',h',w'}^{(l+1)} = \max_{0 \leq j < k, 0 \leq i < k} X_{r,c',(h's+j),(w's+i)}^{(l)} \quad (34)$$

4.1.6 Maxpoollagret bakåtpropagering

Maxpooling saknar vikter och därmed är $\frac{\partial L(W)}{\partial W^{(l)}}$ tom. Det som återstår är bakåtpropageringen av delta-felet. Med hjälp av kedjeregeln kan man dela upp derivatan i två bråk, $\frac{\partial L(W)}{\partial X_{r,c',h',w'}^{(l+1)}}$ och $\frac{\partial X_{r,c',h',w'}^{(l+1)}}{\partial X_{r,c,h,w}^{(l)}} \cdot \frac{\partial L(W)}{\partial X_{r,c',h',w'}^{(l+1)}}$ är den rekursiva delta-delet. $X_{r,c',h',w'}^{(l+1)}$ byts sedan ut mot dess definition enligt ekvation (34): [1] [3] [15]

$$\begin{aligned} \delta_{r,c,h,w}^{(l)} &= \frac{\partial L(W)}{\partial X_{r,c,h,w}^{(l)}} \\ &= \frac{\partial L(W)}{\partial X_{r,c',h',w'}^{(l+1)}} \frac{\partial X_{r,c',h',w'}^{(l+1)}}{\partial X_{r,c,h,w}^{(l)}} \\ &= \delta_{r,c',h',w'} \frac{\partial \max_{0 \leq j < k, 0 \leq i < k} X_{r,c',(h's+j),(w's+i)}^{(l)}}{\partial X_{r,c,h,w}^{(l)}} \end{aligned} \quad (35)$$

Den partiella derivatan i den sista ekvationen kommer vara lika med 1 om $X_{r,c',(h's+j),(w's+i)}^{(l)} = X_{r,c,h,w}^{(l)}$. I annat fall kommer $X_{r,c,h,w}^{(l)}$ inte ha någon påverkan på neuron index (r, c, h, w) i lager $l+1$ och den partiella derivatan blir lika med 0: [1] [3] [15]

$$\delta_{r,c,h,w}^{(l)} = \begin{cases} \delta_{r,c',h',w'} & \text{om } \begin{matrix} h = h's + j, \\ w = w's + i \end{matrix} \\ 0 & \text{i annat fall} \end{cases} \quad (36)$$

Alltså omdiregeras delta-felet till det ansvariga neuronet vars index kommer att behöva hållas i minnet. Om det finns två eller fler sektioner med samma neuron som är ansvarig för framåtpropageringen så kommer delta-felen summeras från samtliga korresponderande sektioners delta-fel. [1] [3] [15]

4.1.7 Batch Normalization framåtpropagering

Utan Batch Normalization (BN) är det svårt att få djupa nätverk att konvergera. Detta är till följd av att en liten ändring till det första lagret kan leda till

en kaskad av förändringar i de senare lagren. I litteraturen kallas detta för *internal covariate shift*. BN försöker att minimera denna *internal covariate shift* genom att med avseende på alla exempel i mini-hopen normalisera varje *feature map* till varje lager. Resultatet är snabbare konvergens och att det tillåter större träningshastigheter. Följaktligen har bearbetningen av flera exempel i taget i en mini-hop en annan praktiska tillämpning än att enbart förskratta uträkningar. [1] [10]

Igen är aktiveringen vid lager l och $l + 1$ $X^{(l)} \in \mathbb{R}^{R \times C \times H \times W}$ respektive $X^{(l+1)} \in \mathbb{R}^{R \times C' \times H' \times W'}$. BN har ingen påverkan på dimensionerna av aktiveringen. [1] [10]

Först beräknas medelvärdena μ_c och varianserna σ_c^2 till varje *feature map* c : [1] [10]

$$\mu_c = \frac{1}{RHW} \sum_{r=0}^{R-1} \sum_{h=0}^{H-1} \sum_{w=0}^{W-1} X_{r,c,h,w}^{(l)} \quad (37)$$

$$\sigma_c^2 = \frac{1}{RHW} \sum_{r=0}^{R-1} \sum_{h=0}^{H-1} \sum_{w=0}^{W-1} (X_{r,c,h,w}^{(l)} - \mu_c)^2 \quad (38)$$

Sedan beräknas den normaliserade aktiveringen \hat{X} . Epsilon används för numerisk stabilitet. [1] [10]

$$\hat{X}_{r,c,h,w} = (X_{r,c,h,w}^{(l)} - \mu_c)(\sigma_c^2)^{-\frac{1}{2}} \quad (39)$$

Sist introduceras 2 vikter, $\gamma_c^{(l)}$ och $\beta_c^{(l)}$, vilka tillåter nätverket att upphäva normaliseringen om nätverket dömmar det att vara användbart. [1] [10]

$$X_{r,c,h,w}^{(l+1)} = \gamma_c^{(l)} \hat{X}_{r,c,h,w} + \beta_c^{(l)} \quad (40)$$

Vid RUNTIME är det dock inte alltid möjligt att beräkna medelvärdet och variansen av mini-hopen på grund av att man oftast enbart vill testa ett exempel i taget. Medelvärdet och variansen för hela populationen måste då räknas ut och användas i stället för de beräknade värdena. Detta kan göras för små DATASETS, men om man arbetar med data som innehåller miljontals exempel är det enklare att uppskatta populationens statistik med hjälp av att updatera ett exponensiellt glidande medelvärde (EWMA) vid varje framåtpropagering: [1] [10]

$$\mu_{EWMA_c} \rightarrow \lambda \mu_c + (1 - \lambda) \mu_{EWMA_c} \quad (41)$$

$$\sigma_{EWMA_c}^2 \rightarrow \lambda \sigma_c^2 + (1 - \lambda) \sigma_{EWMA_c}^2 \quad (42)$$

Där μ_{EWMA_c} och $\sigma_{EWMA_c}^2$ betecknar de exponensiella glidande medelvärdena och λ betecknar dämpfaktorn.

4.1.8 Batch Normalization bakåtpagering

För BN behöver det rekursiva delta-felet $\delta^{(l)}$, derivatan av kostandsfunktionen med avseende på $\gamma_{c'}^{(l)}$ och derivatan av kostandsfunktionen med avseende på $\beta_{c'}^{(l)}$ beräknas. För att beräkna detta krävs något som heter kronecker-deltat, oftast betecknat med $\delta_{i,j}$ men kommer vara betecknat med $I_{i,j}$ i denna rapport på grund av δ används för en annan term. Kronecker-deltat har följande egenskaper: [12] [14]

$$I_{i,j} = \begin{cases} 1 & \text{om } i = j \\ 0 & \text{om } i \neq j \end{cases} \quad (43)$$

$$\frac{\partial a_j}{\partial a_i} = I_{i,j} \quad (44)$$

$$\sum_j a_i I_{i,j} = a_j \quad (45)$$

Först bryts $\frac{\partial L(W)}{\partial X^{(l)}}$ upp i tre bråk och sedan summeras alla partiella derivator likt ekvation (25). [12] [14] Här summeras dessutom R-dimensionen på grund av att aktiveringar från hela mini-hopen har en påverkan på $\delta^{(l)}$.

$$\begin{aligned} \delta_{r,c,h,w}^{(l)} &= \frac{\partial L(W)}{\partial X_{r,c,h,w}^{(l)}} \\ &= \sum_{r'=0}^{R'-1} \sum_{c'=0}^{C'-1} \sum_{h'=0}^{H'-1} \sum_{w'=0}^{W'-1} \frac{\partial L(W)}{\partial X_{r',c',h',w'}^{(l+1)}} \frac{\partial X_{r',c',h',w'}^{(l+1)}}{\partial \hat{X}_{r',c',h',w'}} \frac{\partial \hat{X}_{r',c',h',w'}}{\partial X_{r,c,h,w}^{(l)}} \end{aligned} \quad (46)$$

$\frac{\partial L(W)}{\partial X_{r,c,h,w}^{(l+1)}}$ är det föregående rekursiva delta-felet. $\frac{\partial X_{r',c',h',w'}^{(l+1)}}{\partial \hat{X}_{r',c',h',w'}}$ hittas enkelt på grund av att den är en linjär funktion. [12] [14]

$$\begin{aligned} \frac{\partial X_{r',c',h',w'}^{(l+1)}}{\partial \hat{X}_{r',c',h',w'}} &= \frac{\partial(\gamma_{c'}^{(l)} \hat{X}_{r',c',h',w'} + \beta_{c'}^{(l)})}{\partial \hat{X}_{r',c',h',w'}} \\ &= \gamma_{c'}^{(l)} \end{aligned} \quad (47)$$

För derivatan av den centrerade aktiveringen med avseende på den originella

aktiveringen tillämpas produktregeln: [12] [14]

$$\begin{aligned}\frac{\partial \hat{X}_{r',c',h',w'}}{\partial X_{r,c,h,w}^{(l)}} &= \frac{\partial (X_{r',c',h',w'}^{(l)} - \mu_{c'}) (\sigma_{c'}^2)^{-\frac{1}{2}}}{\partial X_{r,c,h,w}^{(l)}} \\ &= (\sigma_{c'}^2)^{-\frac{1}{2}} \frac{\partial (X_{r',c',h',w'}^{(l)} - \mu_{c'})}{\partial X_{r,c,h,w}^{(l)}} - \frac{1}{2} (X_{r',c',h',w'}^{(l)} - \mu_{c'}) (\sigma_{c'}^2)^{-\frac{3}{2}} \frac{\partial \sigma_{c'}^2}{\partial X_{r,c,h,w}^{(l)}}\end{aligned}\quad (48)$$

Derivatan av den första faktorn med avseende på aktiveringen beräknas med hjälp av ekvationer (43), (44) och (45). [12] [14]

$$\begin{aligned}\frac{\partial (X_{r',c',h',w'}^{(l)} - \mu_{c'})}{\partial X_{r,c,h,w}^{(l)}} &= \frac{\partial (X_{r',c',h',w'}^{(l)} - \frac{1}{RHW} \sum_{r''=0}^{R-1} \sum_{h''=0}^{H-1} \sum_{w''=0}^{W-1} X_{r'',c',h'',w''}^{(l)})}{\partial X_{r,c,h,w}^{(l)}} \\ &= I_{r',r} I_{c',c} I_{h',h} I_{w',w} - \frac{1}{RHW} I_{c',c}\end{aligned}\quad (49)$$

Derivatan av den andra faktorn med avseende på aktiveringen beräknas på ett liknande sätt med hjälp av kedjeregeln och ekvationer (43), (44) och (45). [12] [14]

$$\begin{aligned}\frac{\partial \sigma_{c'}^2}{\partial X_{r,c,h,w}^{(l)}} &= \frac{\partial \frac{1}{RHW} \sum_{r'=0}^{R-1} \sum_{h'=0}^{H-1} \sum_{w'=0}^{W-1} (X_{r',c',h',w'}^{(l)} - \mu_{c'})^2}{\partial X_{r,c,h,w}^{(l)}} \\ &= \frac{1}{RHW} \sum_{r'=0}^{R-1} \sum_{h'=0}^{H-1} \sum_{w'=0}^{W-1} 2(X_{r',c',h',w'}^{(l)} - \mu_{c'}) (I_{r',r} I_{c',c} I_{h',h} I_{w',w} - \frac{1}{RHW} I_{c',c}) \\ &= \frac{2}{RHW} (X_{r,c',h,w}^{(l)} - \mu_{c'}) I_{c',c} - \frac{2}{(RHW)^2} \sum_{r'=0}^{R-1} \sum_{h'=0}^{H-1} \sum_{w'=0}^{W-1} (X_{r',c',h',w'}^{(l)} - \mu_{c'}) \\ &= \frac{2}{RHW} (X_{r,c',h,w}^{(l)} - \mu_{c'}) I_{c',c}\end{aligned}\quad (50)$$

Den sista summan blir lika med noll på grund av att termerna summeras ihop till medelvärdet minus medelvärdet.

När alla komponenter till bakåtpropageringen av delta-felet är beräknade är insättning av ekvation (48), (49) och (50) i ekvation (46) det enda som kvarstår:

$$\begin{aligned}
\delta_{r,c,h,w}^{(l)} &= \sum_{r'=0}^{R-1} \sum_{c'=0}^{C'-1} \sum_{h'=0}^{H'-1} \sum_{w'=0}^{W'-1} \frac{\partial L(W)}{\partial X_{r',c',h',w'}^{(l+1)}} \frac{\partial X_{r',c',h',w'}^{(l+1)}}{\partial \hat{X}_{r',c',h',w'}} \frac{\partial \hat{X}_{r',c',h',w'}}{\partial X_{r,c,h,w}^{(l)}} \\
&= \sum_{r',c',h',w'} \delta_{r',c',h',w'}^{(l+1)} \gamma_{c'}^{(l)} (\sigma_{c'}^2)^{-\frac{1}{2}} (I_{r',r} I_{c',c} I_{h',h} I_{w',w} - \frac{1}{RHW} I_{c',c}) \\
&\quad - \sum_{r',c',h',w'} \delta_{r',c',h',w'}^{(l+1)} \gamma_{c'}^{(l)} \frac{1}{RHW} (X_{r',c',h',w'}^{(l)} - \mu_{c'}) (X_{r,c,h,w}^{(l)} - \mu_c) (\sigma_{c'}^2)^{-\frac{3}{2}} I_{c',c} \\
&= \delta_{r,c,h,w}^{(l+1)} \gamma_c^{(l)} (\sigma_c^2)^{-\frac{1}{2}} - \frac{1}{RHW} \sum_{r',h',w'} \delta_{r',c,h',w'}^{(l+1)} \gamma_c^{(l)} (\sigma_c^2)^{-\frac{1}{2}} \\
&\quad - \frac{1}{RHW} \sum_{r',h',w'} \delta_{r',c,h',w'}^{(l+1)} \gamma_c^{(l)} (X_{r',c,h',w'}^{(l)} - \mu_{c'}) (X_{r,c,h,w}^{(l)} - \mu_c) (\sigma_c^2)^{-\frac{3}{2}} \\
&= \frac{1}{RHW} \gamma_c^{(l)} (\sigma_c^2)^{-\frac{1}{2}} \left(RHW \delta_{r,c,h,w}^{(l+1)} - \sum_{r',h',w'} \delta_{r',c,h',w'}^{(l+1)} \right. \\
&\quad \left. - (X_{r,c,h,w}^{(l)} - \mu_c) (\sigma_c^2)^{-\frac{3}{2}} \sum_{r',h',w'} \delta_{r',c,h',w'}^{(l+1)} (X_{r',c,h',w'}^{(l)} - \mu_{c'}) \right)
\end{aligned} \tag{51}$$

Derivatan av vikterna hittas på ett liknande sätt som ekvation (46) till (51).
[12] [14]

$$\begin{aligned}
\frac{\partial L(W)}{\partial \gamma_c^{(l)}} &= \sum_r \sum_{c'} \sum_{h'} \sum_{w'} \frac{\partial L(W)}{\partial X_{r,c',h',w'}^{(l+1)}} \frac{\partial X_{r,c',h',w'}^{(l+1)}}{\partial \gamma_c^{(l)}} \\
&= \sum_r \sum_{c'} \sum_{h'} \sum_{w'} \delta_{r,c',h',w'}^{(l+1)} \frac{\partial (\gamma_{c'}^{(l)} \hat{X}_{r,c',h',w'} + \beta_{c'}^{(l)})}{\partial \gamma_c^{(l)}} \\
&= \sum_r \sum_{c'} \sum_{h'} \sum_{w'} \delta_{r,c',h',w'}^{(l+1)} \hat{X}_{r,c,h',w'} I_{c',c} \\
&= \sum_r \sum_{h'} \sum_{w'} \delta_{r,c,h',w'}^{(l+1)} \hat{X}_{r,c,h',w'}
\end{aligned} \tag{52}$$

$$\begin{aligned}
\frac{\partial L(W)}{\partial \beta_c^{(l)}} &= \sum_r^{R-1} \sum_{c'}^{C'-1} \sum_{h'}^{H'-1} \sum_{w'}^{W'-1} \frac{\partial L(W)}{\partial X_{r,c',h',w'}^{(l+1)}} \frac{\partial X_{r,c',h',w'}^{(l+1)}}{\partial \beta_c^{(l)}} \\
&= \sum_r^{R-1} \sum_{c'}^{C'-1} \sum_{h'}^{H'-1} \sum_{w'}^{W'-1} \delta_{r,c',h',w'}^{(l+1)} \frac{\partial(\gamma_{c'}^{(l)} \hat{X}_{r,c',h',w'} + \beta_{c'}^{(l)})}{\partial \beta_c^{(l)}} \\
&= \sum_r^{R-1} \sum_{c'}^{C'-1} \sum_{h'}^{H'-1} \sum_{w'}^{W'-1} \delta_{r,c,h',w'}^{(l+1)} I_{c',c} \\
&= \sum_r^{R-1} \sum_{h'}^{H'-1} \sum_{w'}^{W'-1} \delta_{r,c,h',w'}^{(l+1)}
\end{aligned} \tag{53}$$

4.1.9 Softmax framåtpropagering

Funktionen softmax används i det sista lagret för att gränsa aktiveringen till värden i intervallet $[0, 1]$ och har egenskapen att alla prognostiserade värdena i ett exempel från mini-hopen summeras till 1. Om modellen ska klassificera ett objekt som kan vara av C olika klasser kan \hat{y} tolkas som sannolikheten att objektet är av varje klass c ; $0 \leq c < C$. [1]

Inmatningsvärde för softmaxlagret är en matris $X^{(l)} \in \mathbb{R}^{R \times C}$ och producerar en matris $X^{(l+1)} \in \mathbb{R}^{R \times C}$ med samma dimensioner. Softmaxlagret definieras enligt:

$$X_{r,c}^{(l+1)} = \frac{e^{X_{r,c}^{(l)}}}{\sum_{c'=0}^{C-1} e^{X_{r,c'}^{(l)}}} \tag{54}$$

4.1.10 Softmax bakåtpropagering

Softmaxlagret saknar vikter och endast det rekursiva delta-felet beräknas vid bakåtpropagering. Kvotregeln, kedjeregeln och kronecker-deltat I används för att härleda den partiella derivatan. Likt de andra lagren summeras de partiella derivatorna på grund av att aktiveringen för ett neuron i lager $l+1$

är en funktion av alla neuron i lager l : [1] [2] [16]

$$\begin{aligned}
\delta_{r,c}^{(l)} &= \frac{\partial L(W)}{\partial X_{r,c}^{(l)}} \\
&= \sum_{c'=0}^{C-1} \frac{\partial L(W)}{\partial X_{r,c'}^{(l+1)}} \frac{\partial X_{r,c'}^{(l+1)}}{\partial X_{r,c}^{(l)}} \\
&= \sum_{c'=0}^{C-1} \delta_{r,c}^{(l+1)} \frac{(\sum_{c''=0}^{C-1} e^{X_{r,c''}^{(l+1)}})(e^{X_{r,c'}^{(l+1)}})I_{c',c} - (e^{X_{r,c'}^{(l+1)}})(\sum_{c''=0}^{C-1} e^{X_{r,c''}^{(l+1)}} I_{c'',c})}{(\sum_{c''=0}^{C-1} e^{X_{r,c''}^{(l+1)}})^2} \\
&= \sum_{c'=0}^{C-1} \left(\delta_{r,c}^{(l+1)} \frac{(e^{X_{r,c'}^{(l+1)}})I_{c',c}}{\sum_{c''=0}^{C-1} e^{X_{r,c''}^{(l+1)}}} - \frac{(e^{X_{r,c'}^{(l+1)}})(e^{X_{r,c}^{(l+1)}})}{(\sum_{c''=0}^{C-1} e^{X_{r,c''}^{(l+1)}})^2} \right) \\
&= \sum_{c'=0}^{C-1} \delta_{r,c}^{(l+1)} X_{r,c'}^{(l+1)} (I_{c',c} - X_{r,c}^{(l+1)}) \\
&= \delta_{r,c}^{(l+1)} X_{r,c}^{(l+1)} - \sum_{c'=0}^{C-1} \delta_{r,c}^{(l+1)} X_{r,c'}^{(l+1)} X_{r,c}^{(l+1)} \\
&= \delta_{r,c}^{(l+1)} X_{r,c}^{(l+1)} \left(1 - \sum_{c'=0}^{C-1} X_{r,c'}^{(l+1)} \right)
\end{aligned} \tag{55}$$

4.2 Praktiska Tillämpningar

All kod till de exempel på praktiska tillämpningar kan hittas på github: <https://github.com/nikitazozoulenko>

4.2.1 Klassificering av handskrivna siffror

En enkel CNN-modell kan användas för att klassificera handskrivna siffror. För att uppnå detta har MNIST DATASET för handskrivna siffror används. Den består av 60 000 unika exempel för träning och ytterligare 10 000 exempel för validering av modellen. [17]

Låt modellens prognos betecknas med \hat{y} . Aktiveringsfunktionen softmax används i det sista lagret för att gränsa värdena till intervallet $[0, 1]$ och har egenskapen att alla prognostiserade värdena i ett exempel summeras till 1. Följaktligen kan varje värde i \hat{y} tolkas som sannolikheten att bilden är av varje klass. [1]



Figur 9: Tio bilder av handskrivna siffror från MNIST DATASETET. [17]

Input för modellen är en $R \times 1 \times 28 \times 28$ pixelarray där R står för mini-hopstorleken. Modellen prognostiserar tio värden per bild i form av en tensor $\hat{y} \in \mathbb{R}^{R \times C}$, ett värde för varje klass $C = 10$ av siffror. Konstansfunktionen som har minimerats under tränings tiden är funktionen *cross-entropy* betecknat med L . Den verkar på två sannolikhetsfördelningar: De verkliga sannolikheterna y och de prognostiserade sannolikheterna \hat{y} : [1] [2]

$$L(W) = -\frac{1}{R} \sum_{r=0}^{R-1} \sum_{c=0}^{C-1} y_{r,c} \log \hat{y}_{r,c} \quad (56)$$

$$\begin{aligned} \frac{\partial L(W)}{\partial \hat{y}_{r',c'}} &= \frac{\partial \left(-\frac{1}{R} \sum_{r=0}^{R-1} \sum_{c=0}^{C-1} y_{r,c} \log \hat{y}_{r,c} \right)}{\partial \hat{y}_{r',c'}} \\ &= -\frac{1}{R} \sum_{r=0}^{R-1} \sum_{c=0}^{C-1} y_{r,c} \frac{\partial (\log \hat{y}_{r,c})}{\partial \hat{y}_{r',c'}} \\ &= -\frac{1}{R} \sum_{r=0}^{R-1} \sum_{c=0}^{C-1} y_{r,c} \frac{1}{\hat{y}_{r,c}} I_{r,r'} I_{c,c'} \\ &= -\frac{1}{R} \frac{y_{r',c'}}{\hat{y}_{r',c'}} \end{aligned} \quad (57)$$

Där I är kronecker-deltat.

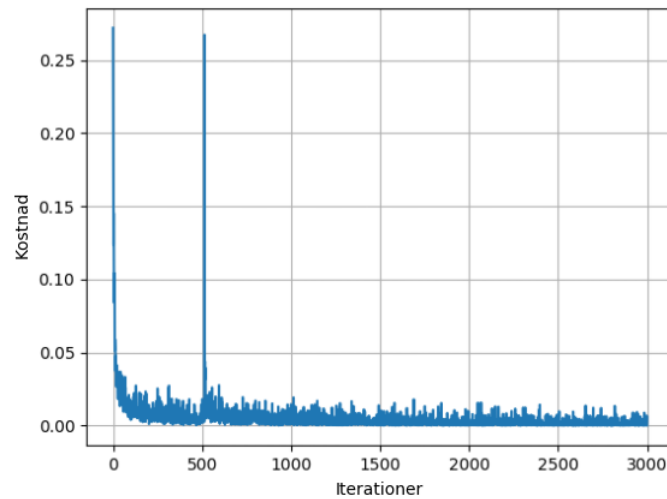
Endast två olika modeller med olika lagerstrukturer prövades på grund av begränsad datakraft. En hopstorlek av 50 användes och 1200 iterationer av framåt- och bakåtpropagering kördes på den egenimplementerade modellen skriven i python. Totalt tog det 50 minuter för varje modell.

Modell 1	Modell 2
Input	Input
BN	BN
conv3 c16 s1	conv3 c16 s1
ReLU	ReLU
BN	BN
conv3 c16 s1	conv3 c16 s1
ReLU	ReLU
BN	BN
conv3 c16 s1	conv3 c16 s1
ReLU	conv3 c16 s1
BN	maxpool2 s2
conv4 c16 s1	ReLU
maxpool2 s2	BN
ReLU	conv4 c16 s1
BN	ReLU
conv4 c10 s1	BN
softmax	conv4 c24 s1
Output	ReLU
	BN
	conv3 c10 s1
	softmax
	Output

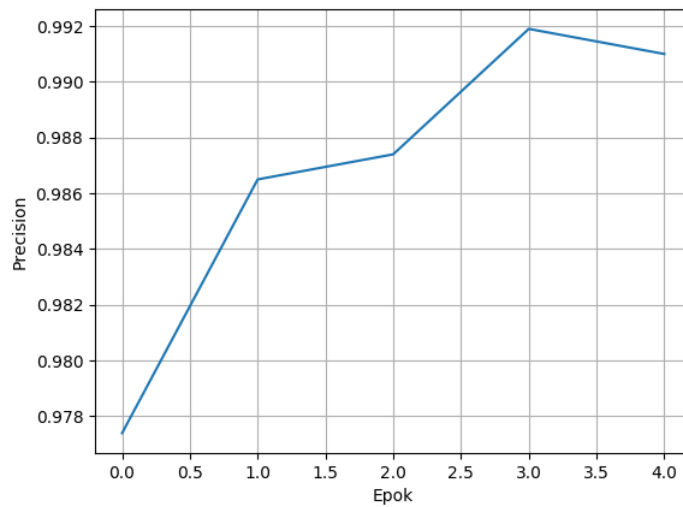
Ett lager är antingen BN, conv (konvolution), maxpool, softmax eller ReLU. Nästinkommande siffra efter lagernamnet är kärnstorleken. C benämner antal kärner vid det lagret och därmed antal kanaler vid nästa lager. Klivet benämns med s .

Modell 1 uppnådde 95.57% precision medan modell 2 uppnådde 98.07% precision. Efter testerna kördes tränades det bättre presterade nätverket, modell 2, på fler iterationer tills validationsprecisionen började avta. Den testades varje 60 000 exempel eller efter varje 600 iterationer med en hopstorlek av 100. Efter varje 60 000 bilder blandades dem för att förbättra inläringen.

Den slutgiltiga precisionen blev 99.2%. Av 10 000 handskrivna siffror lyckades modellen klassifisera 9 919 siffror rätt.



Figur 10: En graf av kostnaden som funktion av antalet iterationer med mini-hopstorleken 100 bilder med modell 2.



Figur 11: En graf av validationsprecisionen som funktion av antalet epoker av 60 000 bilder med modell 2.

4.2.2 Objektdetektering i bilder

4.2.3 Ansiktsigenkänning

Modellen tränas just nu. Jag får resultaten cirka slutet av januari

5 Diskussion

Vet inte vad jag ska diskutera?????

Matten går inte att argumentera med. Det som skulle kunna pratas om är hur jag skulle kunna förbättra modellerna, med det är ju ganska jävla svårt för det här är state of the art (nästan). Jag förstår inte det som folk gör för att förbättra det jag har gjort.

Referenser

- [1] *CS231n: Convolutional Neural Networks for Visual Recognition*. F. Li, A. Karpathy och J. Johnson. Stanford University, föreläsning, vinter 2016.
- [2] *Notes on Backpropagation*. P. Sadowski. University of California Irvine Department of Computer Science.
- [3] *Introduction to Convolutional Neural Networks*. J. Wu. National Key Lab for Novel Software Technology, Nanjing University, Kina. 1 maj, 2017.
- [4] *A guide to convolution arithmetic for deep learning*. V. Dumoulin och F. Visin. FMILA, Université de Montréal. AIRLab, Politecnico di Milano. 24 mars, 2016.
- [5] *High Performance Convolutional Neural Networks for Document Processing*. K. Chellapilla, S. Puri, P. Simard. Tenth International Workshop on Frontiers in Handwriting Recognition. La Baule, Frankrike, Suvisoft. Oktober 2006.
- [6] *Unsupervised Feature Learning and Deep Learning*. Stanford University, Department of Computer Science. URL <http://ufldl.stanford.edu/wiki/>. Senast uppdaterad 31 mars 2013.
- [7] *Scientific Computing 2013, Worksheet 6: Optimization: Gradient and steepest descent*. University of Tartu, Estland. 2013.
- [8] *You only look once: Unified, real-time object detection*. J. Redmon, S. Divvala, R. Girshick, och A. Farhadi. arXiv preprint arXiv:1506.02640, 2015.
- [9] *Deep residual learning for image recognition*. K. He, X. Zhang, S. Ren, och J. Sun. arXiv preprint arXiv:1512.03385, 2015.

- [10] *Batch normalization: Accelerating deep network training by reducing internal covariate shift.* S. Ioffe och C. Szegedy. arXiv preprint arXiv:1502.03167, 2015.
- [11] *Backpropagation In Convolutional Neural Networks.* J. Kafunah. DeepGrid, Organic Deep Learning. URL <http://www.jefkine.com/>. 5 september 2016.
- [12] *What does the gradient flowing through batch normalization looks like?* C. Thorey. Machine Learning Blog. URL <http://cthorey.github.io/>. 28 januari 2016.
- [13] *Note on the implementation of a convolutional neural networks.* C. Thorey. Machine Learning Blog. URL <http://cthorey.github.io/>. 2 februari 2016.
- [14] *Understanding the backward pass through Batch Normalization Layer.* Flaire of Machine Learning URL <https://kratzert.github.io>. 5 september 2016.
- [15] *Convolutional Neural Networks.* A. Gibiansky. URL <http://andrew.gibiansky.com>. 24 februari 2014.
- [16] *Classification and Loss Evaluation - Softmax and Cross Entropy Loss.* P. Dahal. DeepNotes. URL <https://deepnotes.io/softmax-crossentropy>. 24 februari 2014.
- [17] *The MNIST database of handwritten digits* Y. LeCun, C. Cortes och C. Burges. Courant Institute, NYU. Google Labs, New York. Microsoft Research, Redmond. URL <http://yann.lecun.com/exdb/mnist/>. Hämtad 3 november 2017.
- [18] *Pygradsc.* J. Komoroske. URL <https://github.com/joshdk/pygradesc>. 12 oktober 2012.
- [19] *Understanding Convolutional Neural Networks for NLP.* D. Britz. WildML, Artificial Intelligence, Deep Learning, and NLP. 7 november 2015.
- [20] *Understanding Convolutional Neural Networks for NLP.* D. Britz. WildML, Artificial Intelligence, Deep Learning, and NLP. 7 november 2015.
- [21] *Deep learning for complete beginners: convolutional neural networks with keras.* P. Veličković. Camebridge Spark. URL

<https://cambridgespark.com/content>. Senast uppdaterad 20 mars 2017.