Objektdetektering och Ansiktsigenkänning med Konvolutionella Neurala Nätverk

Nikita Zozoulenko

4 november 2017

Abstract

Convolutional Neural Networks have ...

Innehåll

1 Introduktion

1.1 Bakgrund

25 sidor av min rapport är bakgrund ?????

nej men seriöst vet inte vad jag ska skriva här, råd och förslag välkomnas

starkt

1.2 Syfte

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

1.3 Frågeställning

Vad är ett artificiellt neuralt nätverk? Vad är ett Konvolutionellt Neuralt Nätverk? Hur härleds framåt- och bakåtpropageringen i feed-forward neurala nätverk och konvolutionella neurala nätverk? Hur kan modellen tillämpas för att implementera sifferavläsning,ansiktsigenkänning och objektdetektion i bilder?

2 Metod

Majoriteten av tiden gick åt till att... standford, vetenskapliga artiklar batch normalization någonting et al. 2015.... implementera skiten och dubbelkolla med derivatans definition

3 Notation

I denna rapport används notation för bland annat vektorer, matriser och tensorer av högre grad. En tensor av grad 1 är en vektor $x \in \mathbb{R}^H$ och är en radvektor med H element. Matriser M är tensorer av grad 2 sådana att $M \in \mathbb{R}^{H \times W}$. En tensor $X \in \mathbb{R}^{R \times C \times H \times W}$ av grad 4 indexeras med en fyr-tupel (r, c, h, w) där $0 \leq r < R$, $0 \leq c < C$, $0 \leq h < H$ och $0 \leq w < W$. Om $R \times C \times H \times W$ och (r, c, h, w) representerar dimensionerna respektive index för lager l represneterar $R \times C' \times H' \times W'$ och (r, c', h', w') dimensionerna respektive index för nästinkommande lager.

Om X är en matris definieras funktionen $f(X)$ genom att elementvis applicera f på alla matrisens element:

$$f(X) = \begin{bmatrix} f(X_{0,0}) & f(X_{0,1}) & \cdots & f(X_{0,i}) \\ f(X_{1,0}) & f(X_{1,1}) & \cdots & f(X_{1,i}) \\ \vdots & \vdots & \ddots & \vdots \\ f(X_{j,0}) & f(X_{j,1}) & \cdots & f(X_{j,i}) \end{bmatrix} \tag{1}$$

$$\frac{\partial f(X)}{\partial X} = \begin{bmatrix} \frac{\partial f(X_{0,0})}{\partial X_{0,0}} & \frac{\partial f(X_{0,1})}{\partial X_{0,1}} & \cdots & \frac{\partial f(X_{0,i})}{\partial X_{0,i}} \\ \frac{\partial f(X_{1,0})}{\partial X_{1,0}} & \frac{\partial f(X_{1,1})}{\partial X_{1,1}} & \cdots & \frac{\partial f(X_{1,i})}{\partial X_{1,i}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f(X_{j,0})}{\partial X_{j,0}} & \frac{\partial f(X_{j,1})}{\partial X_{j,1}} & \cdots & \frac{\partial f(X_{j,i})}{\partial X_{j,i}} \end{bmatrix} \tag{2}$$

Detta kan generaliseras för en tensor av grad n:

$$\big[f(X)\big]_{i_0, i_1, \cdots, i_{n-1}} = f(X_{i_0, i_1, \cdots, i_{n-1}}) \tag{3}$$

$$\left[\frac{\partial f(X)}{\partial X}\right]_{i_0, i_1, \cdots, i_{n-1}} = \frac{\partial f(X_{i_0, i_1, \cdots, i_{n-1}})}{\partial X_{i_0, i_1, \cdots, i_{n-1}}} \tag{4}$$

Hadamardprodukten betäcknas med $\odot$ och verkar på två tensorer A och B av samma storlek och producerar en tensor C med samma storlek.[1] Elementen i tensorerna multipliceras elementvist :

$$C_{i_0, i_1, \cdots, i_{n-1}} = A_{i_0, i_1, \cdots, i_{n-1}} \odot B_{i_0, i_1, \cdots, i_{n-1}} \tag{5}$$

En tensor av grad n kan ses som en tensor av grad 1 vars element är tensorer av grad n-1. Exempelvis är en vektor en vektor av skalärer och en matris

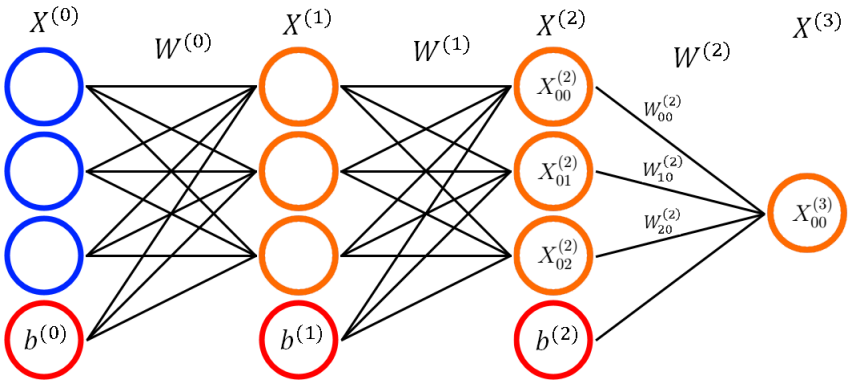
en vektor av vektorer. Funktionen $vec()$ gör om en tensor av grad n till en vektor genom att rekursivt använda den sistnämnda egenskaper av tensorer. [3]

4 Resultat

4.1 Feed-forward Neurala Nätverk

Ett artificellt neuralt nätverk består av ett antal lager neuroner. De är uppbyggda rekursivt så att resultatet av ett lager är inmatningen till nästintilliggande lager. Nervsignalen propageras framåt tills den når det sista lagret. Hur nervceller från ett lager är kopplade till det föregående lagret varierar med vilken typ av neuralt nätverk man väljer. [1]

Den mest grundläggande modellen är så kallade *Multilayer Perceptrons* och har flera namn, bland annat *Fully Connected Cascade (FCC)*, *Feed-forward Neural Network* och *Densly Connected (Dense)*. Modellen består av ett flertal lager av neuroner sådana att resultatet av ett lager matas in som input till nästa lager. Varje neuron i ett lager är kopplade till alla neuroner i nästintillföljande lager. Styrkan av en nervsignals fortplantning till nästa neuron beror på hur stark kopplingen mellan de två neuronerna är. Deras värden vid ett visst lager kallas för aktiveringen vid det lagret. Utöver den vanliga nervsignalsöverföringen appliceras dessutom en aktiveringsfunktion $f(x)$ på samtliga lager elementvis på varje neuron enligt ekvation (3). [1]



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 indatan. Svarta linjer symboliserar vikterna och deras styrka mellan två neuroner.

4.1.1 Framåtprogagering

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

Nätverket kan framställas genom att representera neuronerna och deras kopplingar m.h.a matriser. Låt $X_{ri}^{(l)}$ benämna neuron nummer i i lager l i hop r, $W_{ba}^{(l)}$ styrkan på kopplingen mellan neuron $X_{ra}^{(l)}$ och $X_{rb}^{(l+1)}$ och låt $b^{(l)}$ (efter engelskans *bias*) vara en konstant neuron som är kopplad till alla neuron i lager $l+1$. Output av nätverket kallas för $\hat{y}$ och är värdet av det sista lagret neuroner. Signalöverföringen mellan lager 2 och 3 i figur 1 kan beskrivas matematiskt genom: [1] [6]

$$\begin{aligned} X_{00}^{(3)} &= f(X_{00}^{(2)}W_{00}^{(2)} + X_{01}^{(2)}W_{10}^{(2)} + X_{02}^{(2)}W_{20}^{(2)} + b^{(2)}) \\ &= f(X^{(2)}W^{(2)} + b^{(2)}) \end{aligned} \tag{6}$$

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 $b^{(2)} \in \mathbb{R}^{1}$.

Mer generellt kan ett helt neuralt nätverk beskrivas med följande rekursiva formel: [1] [6]

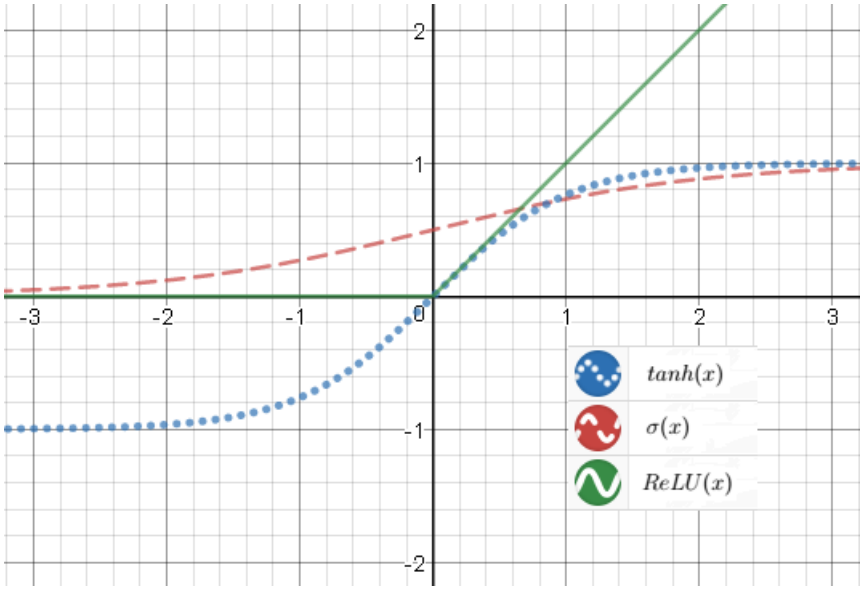
$$X^{(l+1)} = f(X^{(l)}W^{(l)} + b^{(l)}) \tag{7}$$

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 bakåtpropagering. Utan aktiveringsfunktioner skulle hela modellen vara en linjär transformation. Genom att använda olinjära funktioner kan nätverket lära sig olinjära samband. Definitionerna av funktionerna ges av: [1]

$$ReLU(x) = \begin{cases} 0 & \text{om } x < 0 \\ x & \text{om } x \geq 0 \end{cases} \tag{8}$$

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{9}$$

$$tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{10}$$



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

För att derivera $ReLU$ deriveras varje enskilda fall:

$$\frac{\partial(ReLU(x))}{\partial x} = \begin{cases} 0 & \text{om } x < 0 \\ 1 & \text{om } x \geq 0 \end{cases} \tag{11}$$

Notera att:

$$\lim_{x \to 0^+} ReLU(x) = 1 \tag{12}$$

$$\lim_{x \to 0^-} ReLU(x) = 0 \tag{13}$$

Trots att derivatan av ReLU inte är definierad i punkten $x = 0$ sätts $\frac{\partial(ReLU(x))}{\partial x}|_{x=0}$ vara lika med 0 eller 1 utan att några problem tillkommer. [1]

Kvotregeln tillämpas för att härleda derivatan av σ och $tanh$:

$$\begin{aligned} \frac{\partial \sigma(x)}{\partial x} &= \frac{e^{-x}}{(1+e^{-x})^2} \\ &= \frac{1}{1+e^{-x}}\left(\frac{1+e^{-x}}{1+e^{-x}} - \frac{1}{1+e^{-x}}\right) \\ &= \sigma(x)(1-\sigma(x)) \end{aligned} \tag{14}$$

$$\begin{aligned}\frac{\partial(tanh(x))}{\partial x} &= \frac{(e^x + e^{-x})(e^x + e^{-x}) - (e^x - e^{-x})(e^x - e^{-x})}{(e^x + e^{-x})^2} \\ &= 1 - \frac{(e^x - e^{-x})^2}{(e^x + e^{-x})^2} \\ &= 1 - tanh^2(x)\end{aligned} \tag{15}$$

4.1.2 Bakåtpropagation

Given en input X, vill du prognostisera ett värde $\hat{y}$ Detta värde ska vara så likt THE GROUND TRUTH y som möjligt. När man först initialiserar modellen kommer vikterna $W^{(l)}$ vara slumpade och nätverkets prognos kommer inte efterlikna det eftersökta 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 variablerna $W^{(0)}$, $W^{(1)}$, ..., $W^{(l)}$, $b^{(0)}$, $b^{(1)}$, ..., $b^{(l)}$ 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 liten 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]

$$\begin{aligned}L(W, b) &= ||\hat{y} - y_r||^2 \\ &= ||f(f(f(XW^{(0)} + b^{(0)})W^{(1)} + b^{(1)})W^{(2)} + b^{(2)}) - y||^2\end{aligned} \tag{16}$$

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

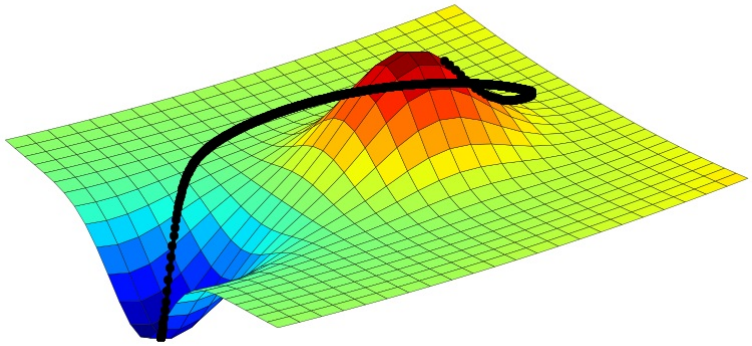
$$\nabla L(\theta) : \mathbb{R}^n \to \mathbb{R}^n \tag{17}$$

$$\nabla L(\theta) = \begin{pmatrix} \frac{\partial L(\theta)}{\partial \theta^{(0)}} & \frac{\partial L(\theta)}{\partial \theta^{(1)}} & \dots & \frac{\partial L(\theta)}{\partial \theta^{(n-1)}} \end{pmatrix} \tag{18}$$

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 Gradiant Descent (SGD)* och använder hyperparametern α för att beteckna träningshastigheten: [7] [3] [6]

$$\frac{\partial L(\theta)}{\partial \theta^{(l)}} = \nabla_{\theta^{(l)}} L(\theta) \tag{19}$$

$$\theta^{(l)} \rightarrow \theta^{(l)} - \alpha \frac{\partial L(\theta)}{\partial \theta^{(l)}} \tag{20}$$



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

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

$$\frac{\partial L(\theta)}{\partial \theta^{(i)}} = \frac{L(\theta^{(0)}, ..., \theta^{(i)} + h, ..., \theta^{(n-1)}) - L(\theta)}{h} \tag{21}$$

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 skulle behöva en enorm datorkraft. Istället kan man tillämpa regler för matriskalkyl och kedjeregeln för att effektivt beräkna de partiella derivatorna. Genom att använda ett kedjeliknande argument kan derivatorna uttryckas som: [1] [3]

$$\frac{\partial L(\theta)}{\partial (vec(W^{(l)})^T)} = \frac{\partial L(\theta)}{\partial (vec(X^{(l+1)})^T)} \frac{\partial (vec(X^{(l+1)}))}{\partial (vec(W^{(l)})^T)} \tag{22}$$

$$\frac{\partial L(\theta)}{\partial (vec(b^{(l)})^T)} = \frac{\partial L(\theta)}{\partial (vec(X^{(l+1)})^T)} \frac{\partial (vec(X^{(l+1)}))}{\partial (vec(b^{(l)})^T)} \tag{23}$$

Där

$$\frac{\partial L(\theta)}{\partial (vec(X^{(l)})^T)} = \frac{\partial L(\theta)}{\partial (vec(X^{(l+1)})^T)} \frac{\partial (vec(X^{(l+1)}))}{\partial (vec(X^{(l)})^T)} \tag{24}$$

För att $\frac{\partial X^{(l+1)}}{\partial X^{(l)}}$, $\frac{\partial X^{(l+1)}}{\partial W^{(l)}}$, $\frac{\partial X^{(l+1)}}{\partial b^{(l)}}$ och derivatan av L med avseende på det sista lagret är möjliga att beräknas algebraiskt är det därför möjligt att rekursivt beräkna gradienten genom att bakåtpropagera i nätverket från outputlagret. Värdena från det nästkommande lagret används för att beräkna värdena för det föregående lagret. [3]

Betrakta ekvation (7). Om L är en L2 kostandsfunktion beräknas de partiella derivatorna enligt: [1] [3]

$$\frac{\partial L(\theta)}{\partial \hat{y}} = 2||\hat{y} - y|| \tag{25}$$

I följande ekvationer utnyttjas $\frac{\partial AB}{\partial B} = A^T$ och $\frac{\partial AB}{\partial A} = B^T$ tillsammans med kedjeregeln.

$$\begin{aligned} \frac{\partial X^{(l+1)}}{\partial X^{(l)}} &= \frac{\partial f(X^{(l)}W^{(l)} + b^{(l)})}{\partial X^{(l)}} \\ &= f'(X^{(l)}W^{(l)} + b^{(l)})W^{(l)^T} \end{aligned} \tag{26}$$

$$\begin{aligned} \frac{\partial X^{(l+1)}}{\partial W^{(l)}} &= \frac{\partial f(X^{(l)}W^{(l)} + b^{(l)})}{\partial X^{(l)}} \\ &= X^{(l)^T} f'(X^{(l)}W^{(l)} + b^{(l)}) \end{aligned} \tag{27}$$

$$\begin{aligned} \frac{\partial X^{(l+1)}}{\partial b^{(l)}} &= \frac{\partial f(X^{(l)}W^{(l)} + b^{(l)})}{\partial X^{(l)}} \\ &= f'(X^{(l)}W^{(l)} + b^{(l)}) \end{aligned} \tag{28}$$

Här kan man se varför aktiveringsfunktionerna behöver vara deriverbara.

I praktiken beräknas de partiella derivatorna med hjälp av en rekursiv formel. Låt: [1] [6] [3]

$$\delta^{(l)} = \frac{\partial L(\theta)}{\partial X^{(l)}} \tag{29}$$

Det (också kallat delta-felet) är den bakåtpropagerade nervsignalen upp till lager l. Om l_{sista} är det sista lagret definieras delta-felen och gradienten enligt följande formler för en L2 kostandsfunktion: [1] [6] [3]

$$\delta^{(l_{sista})} = 2||\hat{y} - y|| \tag{30}$$

$$\begin{aligned}\delta^{(l)} &= \frac{\partial L(\theta)}{\partial X^{(l)}} \\ &= \frac{\partial L(\theta)}{\partial X^{(l+1)}}\frac{\partial X^{(l+1)}}{\partial X^{(l)}} \\ &= \left(\delta^{(l+1)} \odot f'(X^{(l)}W^{(l)} + b^{(l)})\right) W^{(l+1)^T}\end{aligned} \tag{31}$$

$$\begin{aligned}\frac{\partial L(\theta)}{\partial W^{(l)}} &= \frac{\partial L(\theta)}{\partial X^{(l+1)}}\frac{\partial X^{(l+1)}}{\partial W^{(l)}} \\ &= X^{(l)^T}\left(\delta^{(l+1)} \odot f'(X^{(l)}W^{(l)} + b^{(l)})\right)\end{aligned} \tag{32}$$

$$\begin{aligned}\frac{\partial L(\theta)}{\partial b^{(l)}} &= \frac{\partial L(\theta)}{\partial X^{(l+1)}}\frac{\partial X^{(l+1)}}{\partial b^{(l)}} \\ &= \delta^{(l+1)} \odot f'(X^{(l)}W^{(l)} + b^{(l)})\end{aligned} \tag{33}$$

Två implementationer av ett feedforward neuralt nätverk kan hittas på github i python och C++: `https://github.com/nikitazozoulenko`

4.2 Konvolutionella Neuala Netvärk

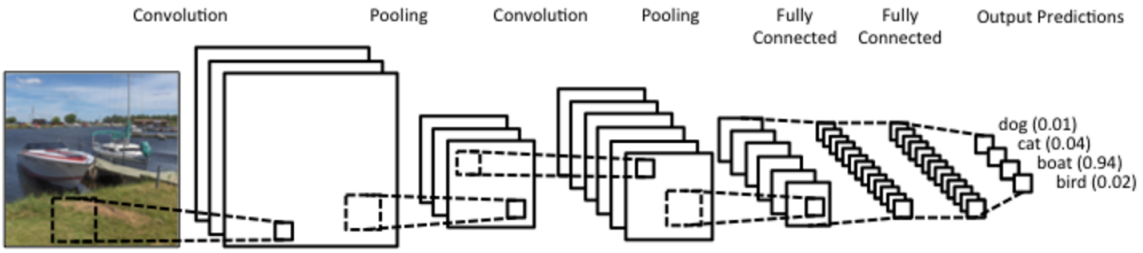
När människor vill identifiera någonting i en bild så letar vi efter vissa karakteristiska drag objektet har. En hund består exempelvis av en kropp, ett huvud och fyra ben. Kroppsdelarna består sedan själva av grundläggande geometriska former som i sig självt är kombinationer av kanter och linjer. Dessutom 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.Det är just detta som är principen bakom *Konvolutionella Neurala Nätverk (CNN)*: Genom så kallade *konvolutioner* kunna extrahera dessa karakteristiska drag. Nätverket lär sig ett antal filter väldigt små filter som den applicerar på en delmängd av bilden genom att filtret sammanrullar över hela bilden. Värdet av filtret över en delmängd av bilden blir aktiveringen av ett neuron i nästa lager. [1]

Till skillnad från *FCC* är neuronerna i ett *CNN* bara kopplade till närliggande neuroner 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 identifiera kanter och linjer medan de senare

Figur 4: Resultatet av ett en kärna för horizontell och vertikal kantdetektering har sammanrullat över en bild av en katt.

lagren lagren känner igen olika former och till sist känna igen ansikten eller object i sista lagert. [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. För ett *FCC* användes en matris för att representera neuronerna. I ett *CNN* är en tensor $X^{(l)} \in \mathbb{R}^{R \times C \times H \times W}$ av grad 4 aktiveringen vid lager l. 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 benämns med C. R står för hopstorlek då man bearbetar R exempel i taget i en så kallad mini-hop. 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. [1] [3]



Figur 5: En illustration av ett konvolutionellt neuralt nätverk. [19]

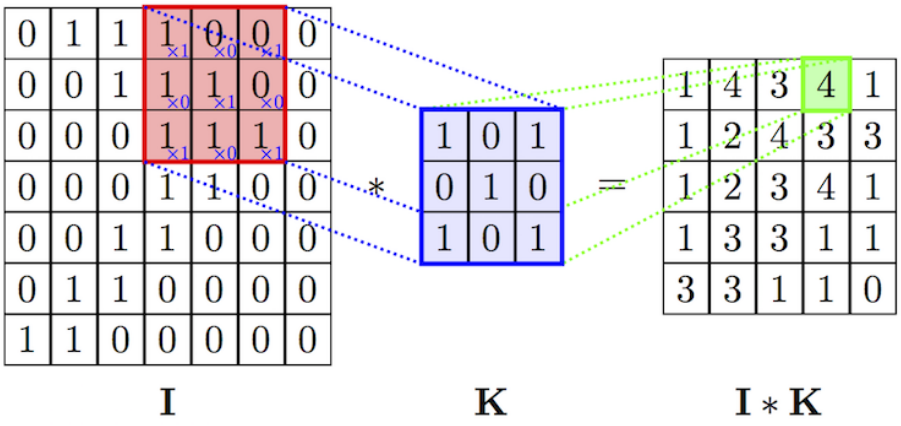
För att en konvolution är en lokal operator används CNNs för data som innehåller lokalt sammanhängande samband, exempelvis bilder eller ljud. Om det är en bild som bearbetas har det första lagrets aktivering $C = 3$ kanaler, en för varje RGB-kanal, och en bredd och höjd lika med bildens

bredd och höjd i pixlar. [1] [3]

4.2.1 Konvolutionslagret framåtpropagering

Ett konvolutionslager består av ett antal vikter kallade *kärnor (kernels)* eller *masker (masks)*, representerade av en tensor av grad 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. Konvolutionsoperatorn 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]

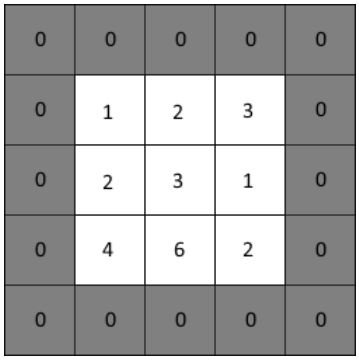
En feature map i lager $l + 1$ är resultatet av att en kärna med dimensioner $1 \times \times C \times k_h \times k_W$ har sammanrullat över aktiveringen av det föregående lagret. C' är antalet kärnor och blir dessutom antalet feature maps nästa lager har. [1] [3]

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 tensorns 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]

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]

$$W' = \frac{W - k_W + 2p}{s} + 1 \tag{34}$$

$$H' = \frac{H - k_H + 2p}{s} + 1 \tag{35}$$



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

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

$$w = sw' \tag{36}$$

$$h = sh' \tag{37}$$

$$\begin{aligned} \left[X^{(l+1)}\right]_{r,c',h',w'} &= X^{(l)}_{r,c',h',w'} * W^{(l)}_{c'} \\ &= \sum_{c=0}^{C-1} \sum_{j=0}^{k_H-1} \sum_{i=0}^{k_W-1} X^{(l)}_{r,c,h'+j,w'+i} W^{(l)}_{c',c,j,i} \end{aligned} \tag{38}$$

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

I praktiken brukar konvolutioner implementeras med hjälp av funktionerna $row2im$ och $im2row$ vilka lämnas till läsaren att läsa på om om han eller hon vill optimera hur snabbt konvolutionen beräknas. [1] [3] [4]

4.2.2 Konvolutionslagret bakåtpropagering

Bakåtpropegeringen förstås bäst genom att algebraiskt härleda den.

Bakåtpropageringen av det rekursiva delta-felet $\frac{\partial L(W)}{\partial X^{(l+1)}_{r,c',h',w'}}$ räkas ut med hjälp av kedjeregeln. Derivatan kan inte endast delas upp i $\frac{\partial L(W)}{\partial X^{(l+1)}_{r,c',h',w'}}$ och $\frac{\partial X^{(l+1)}_{r,c',h',w'}}{\partial X^{(l)}_{r,c,h,w}}$, utan alla derivator måste summeras på grund av att det är mer än ett neuron

som är ansvarig för framåtpropageringen. $X^{(l+1)}_{r,c',h',w'}$ byts sedan ut mot dess definition enligt ekvation (38). [3] [11] [13] [15]

$$\begin{aligned}\delta^{(l)}_{r,c,h,w} &= \frac{\partial L(W)}{\partial X^{(l)}_{r,c,h,w}} \\ &= \sum_{c'=0}^{C'-1}\sum_{h'=0}^{H'-1}\sum_{w'=0}^{W'-1} \frac{\partial L(W)}{\partial X^{(l+1)}_{r,c',h',w'}} \frac{\partial X^{(l+1)}_{r,c',h',w'}}{\partial X^{(l)}_{r,c,h,w}} \\ &= \sum_{c'=0}^{C'-1}\sum_{h'=0}^{H'-1}\sum_{w'=0}^{W'-1} \delta^{(l+1)}_{r,c',h',w'} \frac{\partial \sum_{c=0}^{C-1}\sum_{j=0}^{k_H-1}\sum_{i=0}^{k_W-1} X^{(l)}_{r,c,h'+j,w'+i} W^{(l+1)}_{c',c,j,i}}{\partial X^{(l)}_{r,c,h,w}}\end{aligned} \tag{39}$$

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

$$\begin{aligned}\sum_{c'}^{C'-1}\sum_{h'=0}^{H'-1}\sum_{w'=0}^{W'-1} \delta^{(l+1)}_{r,c',h',w'} \frac{\partial \sum_{c=0}^{C-1}\sum_{j=0}^{k_H-1}\sum_{i=0}^{k_W-1} X^{(l)}_{r,c,h'+j,w'+i} W^{(l+1)}_{c',c,j,i}}{\partial X^{(l)}_{r,c,h,w}} \\ = \sum_{c'=0}^{C'-1}\sum_{h'=0}^{H'-1}\sum_{w'=0}^{W'-1} \delta^{(l+1)}_{r,c',h',w'} W^{(l+1)}_{c',c,j,i} \\ = \sum_{c'=0}^{C'-1}\sum_{h'=0}^{H'-1}\sum_{w'=0}^{W'-1} W^{(l+1)}_{c',c,(h-h'),(w-w')} \delta^{(l+1)}_{r,c',h',w'}\end{aligned} \tag{40}$$

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 betäcknas med funktionen $rot()$. [11] [13] [15]

$$\delta^{(l)}_{r,c,h,w} = \sum_{c'=0}^{C'-1} rot(W^{(l+1)}_{c',c,h,w}) * \delta^{(l+1)}_{r,c'} \tag{41}$$

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.

Den partiella derivatan av kostandsfunktionen med avseende på vikterna hittas på ett liknande sätt: [11] [13] [15]

$$\begin{aligned}
\frac{\partial L(W)}{\partial W^{(l)}_{c',c,k_H,k_W}} &= \frac{1}{R}\sum_{r=0}^{R-1}\sum_{c'=0}^{C'-1}\sum_{h'=0}^{H'-1}\sum_{w'=0}^{W'-1}\frac{\partial L(W)}{\partial X^{(l+1)}_{r,c',h',w'}}\frac{\partial X^{(l+1)}_{r,c',h',w'}}{\partial W^{(l)}_{r,c,h,w}} \\
&= \frac{1}{R}\sum_{r=0}^{R-1}\sum_{c'=0}^{C'-1}\sum_{h'=0}^{H'-1}\sum_{w'=0}^{W'-1}\delta^{(l+1)}_{r,c',h',w'}\frac{\partial \sum_{c=0}^{C-1}\sum_{j=0}^{k_H-1}\sum_{i=0}^{k_W-1} X^{(l)}_{r,c,h'+j,w'+i}W^{(l)}_{c',c,j,i}}{\partial W^{(l)}_{c',c,k_H,k_W}} \\
&= \frac{1}{R}\sum_{r=0}^{R-1}\sum_{c'=0}^{C'-1}\sum_{h'=0}^{H'-1}\sum_{w'=0}^{W'-1} X^{(l)}_{r,c,h'+k_H,w'+k_W}\delta^{(l+1)}_{r,c',h',w'} \\
&= \frac{1}{R}\sum_{r=0}^{R-1}\sum_{c'=0}^{C'-1} X^{(l)}_{r,c,k_H,k_W} * \delta^{(l+1)}_{r,c'}
\end{aligned} \tag{42}$$

Summan av alla exempel i hopen och division med hopstorleken är ett direkt resultat av att man bearbetar flera exempel i taget. Man beräknar medelvärdet av alla gradienter av alla exemepel i hopen. [1]

4.2.3 Aktiveringsfunktionslager framåtpropagering

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

$$X^{(l+1)}_{r,c,h,w} = f(X^{(l)}_{r,c,h,w}) \tag{43}$$

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

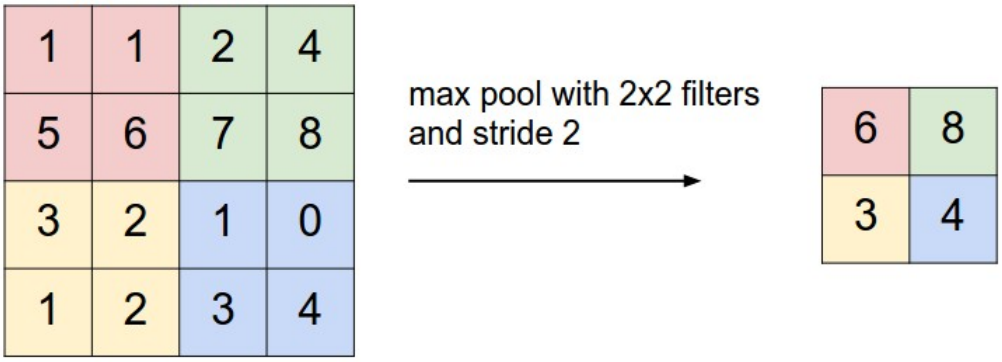
4.2.4 Aktiveringsfunktionslager bakåtpropagering

Aktiveringsfunktioner har inga parametrar som ska optimeras och sålades är $W^{(l)}$ och $\frac{\partial L(W)}{\partial W^{(l)}}$ tomma. Bakåtpropageringen av nervsignalen härleds med hjälp av kedjeregeln och ekvation (4): [1] [3]

$$\begin{aligned}\delta^{(l)}_{r,c,h,w} &= \frac{\partial L(W)}{\partial X^{(l)}_{r,c,h,w}} \\ &= \frac{\partial L(W)}{\partial X^{(l+1)}_{r,c,h,w}} \frac{\partial X^{(l+1)}_{r,c,h,w}}{\partial X^{(l)}_{r,c,h,w}} \\ &= \delta^{(l+1)}_{r,c,h,w} f'(X^{(l)}_{r,c,h,w})\end{aligned} \tag{44}$$

4.2.5 Maxpoollagret 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 har inga 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 samanrullar 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]

$$W' = \frac{W-k}{s} + 1 \tag{45}$$

$$H' = \frac{H-k}{s} + 1 \tag{46}$$

$$C' = C \tag{47}$$

Antal kanaler förblir konstant.

Matematiskt beskrivs maxpoollagret genom: [1] [3]

$$X^{(l+1)}_{r,c',h',w'} = \max_{0\leq j<k,\ 0\leq i<k} X^{(l)}_{r,c',(h's+j),(w's+i)} \tag{48}$$

4.2.6 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, $\frac{\partial L(W)}{\partial X^{(l+1)}_{r,c',h',w'}}$ och $\frac{\partial X^{(l+1)}_{r,c',h',w'}}{\partial X^{(l)}_{r,c,h,w}}$. $\frac{\partial L(W)}{\partial X^{(l+1)}_{r,c',h',w'}}$ är den rekursiva delta-delet. $X^{(l+1)}_{r,c',h',w'}$ byts sedan ut mot dess definition enligt ekvation (48): [1] [3] [15]

$$\begin{aligned}
\delta^{(l)}_{r,c,h,w} &= \frac{\partial L(W)}{\partial X^{(l)}_{r,c,h,w}} \\
&= \frac{\partial L(W)}{\partial X^{(l+1)}_{r,c',h',w'}} \frac{\partial X^{(l+1)}_{r,c',h',w'}}{\partial X^{(l)}_{r,c,h,w}} \\
&= \delta_{r,c',h',w'} \frac{\partial \max\limits_{0\leq j<k, 0\leq i<k} X^{(l)}_{r,c',(h's+j),(w's+i)}}{\partial X^{(l)}_{r,c,h,w}}
\end{aligned} \tag{49}$$

Den partiella derivatan i den sista ekvationen kommer vara lika med 1 om $X^{(l)}_{r,c',(h's+j),(w's+i)} = X^{(l)}_{r,c,h,w}$. I annat fall kommer $X^{(l)}_{r,c,h,w}$ 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]

$$\delta^{(l)}_{r,c,h,w} = \begin{cases} \delta_{r,c,h',w'} & \text{om } \begin{matrix} h = h's + j, \\ w = w's + i \end{matrix} \\ 0 & \text{i annat fall} \end{cases} \tag{50}$$

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]

4.2.7 Batch Normalization framåtpropagering

Utan Batch Normalization (BN) är det svårt att få djupa nätverk att divergera. 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 divergens och att det tillåter större träningshastigheter. Alltså har att man bearbetar flera exemepl i taget i en mini-hop en annan praktiska tillämpning än att försnabba 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]

$$\mu_c = \frac{1}{RHW}\sum_{r=0}^{R-1}\sum_{h=0}^{H-1}\sum_{w=0}^{W-1} X_{r,c,h,w}^{(l)} \tag{51}$$

$$\sigma_c^2 = \frac{1}{RHW}\sum_{r=0}^{R-1}\sum_{h=0}^{H-1}\sum_{w=0}^{W-1} (X_{r,c,h,w}^{(l)} - \mu_c)^2 \tag{52}$$

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

$$\hat{X}_{r,c,h,w} = (X_{r,c,h,w}^{(l)} - \mu_c)(\sigma_c^2 + \epsilon)^{-\frac{1}{2}} \tag{53}$$

Sist introduceras 2 vikter, $\gamma_{c'}^{(l)}$ och $\beta_{c'}^{(l)}$, vilka tillåter nätverket att upphäva normaliseringen om nätverket dömmer det att vara användbart. [1] [10]

$$X_{r,c,h,w}^{(l+1)} = \gamma_c^{(l)}\hat{X}_{r,c,h,w} + \beta_c^{(l)} \tag{54}$$

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]

$$\mu_{EWMA_c} \to \lambda\mu_c + (1-\lambda)\mu_{EWMA_c} \tag{55}$$

$$\sigma_{EWMA_c}^2 \to \lambda\sigma_c^2 + (1-\lambda)\sigma_{EWMA_c}^2 \tag{56}$$

Där μ_{EWMA_c} och $\sigma^2_{EWMA_c}$ betäcknar de exponensiella glidande medelvärdena och λ betäcknar dämpfaktorn.

4.2.8 Batch Normalization bakåtpropagering

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 betäcknat med $\delta_{i,j}$ men kommer vara betäcknat med $I_{i,j}$ i denna rapport på grund av δ används för en annan term. Kronecker-deltat har följande egenskaper: [12] [14]

$$I_{i,j} = \begin{cases} 1 & \text{om } i = j \\ 0 & \text{om } i \neq j \end{cases} \tag{57}$$

$$\frac{\partial a_j}{\partial a_i} = I_{i,j} \tag{58}$$

$$\sum_j a_i I_{i,j} = a_j \tag{59}$$

Först bryts $\frac{\partial L(W)}{\partial X^{(l)}}$ upp i tre bråk och sedan summeras alla partiella derivator likt ekvation (39). [12] [14] Här summeras dessutom R-dimensionen på grund av att aktiveringar från hela mini-hopen har en påverkan på $\delta^{(l)}$.

$$\begin{aligned} \delta^{(l)}_{r,c,h,w} &= \frac{\partial L(W)}{\partial X^{(l)}_{r,c,h,w}} \\ &= \sum_{r'=0}^{R'-1} \sum_{c'=0}^{C'-1} \sum_{h'=0}^{H'-1} \sum_{w'=0}^{W'-1} \frac{\partial L(W)}{\partial X^{(l+1)}_{r',c',h',w'}} \frac{\partial X^{(l+1)}_{r',c',h',w'}}{\partial \hat{X}_{r',c',h',w'}} \frac{\partial \hat{X}_{r',c',h',w'}}{\partial X^{(l)}_{r,c,h,w}} \end{aligned} \tag{60}$$

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

$$\begin{aligned} \frac{\partial X^{(l+1)}_{r',c',h',w'}}{\partial \hat{X}_{r',c',h',w'}} &= \frac{\partial(\gamma^{(l)}_{c'} \hat{X}_{r',c',h',w'} + \beta^{(l)}_{c'})}{\partial \hat{X}_{r',c',h',w'}} \\ &= \gamma^{(l)}_{c'} \end{aligned} \tag{61}$$

För derivatan av den centrerade aktiveringen med avseende på den originella aktiveringen tillämpas produktregeln: [12] [14]

$$\begin{aligned}\frac{\partial \hat{X}_{r',c',h',w'}}{\partial X^{(l)}_{r,c,h,w}} &= \frac{\partial (X^{(l)}_{r',c',h',w'} - \mu_{c'})(\sigma^2_{c'} + \epsilon)^{-\frac{1}{2}}}{\partial X^{(l)}_{r,c,h,w}} \\ &= (\sigma^2_{c'} + \epsilon)^{-\frac{1}{2}} \frac{\partial (X^{(l)}_{r',c',h',w'} - \mu_{c'})}{\partial X^{(l)}_{r,c,h,w}} - \frac{1}{2}(X^{(l)}_{r',c',h',w'} - \mu_c)(\sigma^2_{c'} + \epsilon)^{-\frac{3}{2}} \frac{\partial \sigma^2_{c'}}{\partial X^{(l)}_{r,c,h,w}}\end{aligned} \tag{62}$$

Derivatan av den första faktorn med avseende på aktiveringen beräknas med hjälp av ekvationer (57), (58) och (59). [12] [14]

$$\begin{aligned}\frac{\partial (X^{(l)}_{r',c',h',w'} - \mu_{c'})}{\partial X^{(l)}_{r,c,h,w}} &= \frac{\partial (X^{(l)}_{r',c',h',w'} - \frac{1}{RHW} \sum_{r''=0}^{R-1} \sum_{h''=0}^{H-1} \sum_{w''=0}^{W-1} X^{(l)}_{r'',c',h'',w''})}{\partial X^{(l)}_{r,c,h,w}} \\ &= I_{r',r} I_{c',c} I_{h',h} I_{w',w} - \frac{1}{RHW} I_{c',c}\end{aligned} \tag{63}$$

Derivatan av den andra faktorn med avseende på aktiveringen beräknas på ett liknande sätt med hjälp av kedjeregeln och ekvationer (57), (58) och (59). [12] [14]

$$\begin{aligned}\frac{\partial \sigma^2_{c'}}{\partial X^{(l)}_{r,c,h,w}} &= \frac{\partial \frac{1}{RHW} \sum_{r'=0}^{R-1} \sum_{h'=0}^{H-1} \sum_{w'=0}^{W-1} (X^{(l)}_{r',c',h',w'} - \mu_{c'})^2}{\partial X^{(l)}_{r,c,h,w}} \\ &= \frac{1}{RHW} \sum_{r'=0}^{R-1} \sum_{h'=0}^{H-1} \sum_{w'=0}^{W-1} 2(X^{(l)}_{r',c',h',w'} - \mu_{c'})(I_{r',r} I_{c',c} I_{h',h} I_{w',w} - \frac{1}{RHW} I_{c',c}) \\ &= \frac{2}{RHW}(X^{(l)}_{r,c',h,w} - \mu_{c'}) I_{c',c} - \frac{2}{(RHW)^2} \sum_{r'=0}^{R-1} \sum_{h'=0}^{H-1} \sum_{w'=0}^{W-1} (X^{(l)}_{r',c,h',w'} - \mu_c) \\ &= \frac{2}{RHW}(X^{(l)}_{r,c',h,w} - \mu_{c'}) I_{c',c}\end{aligned} \tag{64}$$

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 (62), (63) och (64) i ekvation (60) det enda som kvarstår:

$$
\begin{aligned}
\delta^{(l)}_{r,c,h,w} &= \sum_{r'=0}^{R'-1}\sum_{c'=0}^{C'-1}\sum_{h'=0}^{H'-1}\sum_{w'=0}^{W'-1}\frac{\partial L(W)}{\partial X^{(l+1)}_{r',c',h',w'}}\frac{\partial X^{(l+1)}_{r',c',h',w'}}{\partial \hat{X}_{r',c',h',w'}}\frac{\partial \hat{X}_{r',c',h',w'}}{\partial X^{(l)}_{r,c,h,w}} \\
&= \sum_{r',c',h',w'}\delta^{(l+1)}_{r',c',h',w'}\gamma^{(l)}_{c'}(\sigma^2_{c'}+\epsilon)^{-\frac{1}{2}}(I_{r',r}I_{c',c}I_{h',h}I_{w',w}-\frac{1}{RHW}I_{c',c}) \\
&- \sum_{r',c',h',w'}\delta^{(l+1)}_{r',c',h',w'}\gamma^{(l)}_{c'}\frac{1}{RHW}(X^{(l)}_{r',c',h',w'}-\mu_{c'})(X^{(l)}_{r,c',h,w}-\mu_{c'})(\sigma^2_{c'}+\epsilon)^{-\frac{3}{2}}I_{c',c} \\
&= \delta^{(l+1)}_{r,c,h,w}\gamma^{(l)}_{c}(\sigma^2_{c}+\epsilon)^{-\frac{1}{2}} - \frac{1}{RHW}\sum_{r',h',w'}\delta^{(l+1)}_{r',c,h',w'}\gamma^{(l)}_{c}(\sigma^2_{c}+\epsilon)^{-\frac{1}{2}} \\
&- \frac{1}{RHW}\sum_{r',h',w'}\delta^{(l+1)}_{r',c,h',w'}\gamma^{(l)}_{c}(X^{(l)}_{r',c,h',w'}-\mu_{c'})(X^{(l)}_{r,c,h,w}-\mu_{c})(\sigma^2_{c}+\epsilon)^{-\frac{3}{2}} \\
&= \frac{1}{RHW}\gamma^{(l)}_{c}(\sigma^2_{c}+\epsilon)^{-\frac{1}{2}}\Bigg(RHW\delta^{(l+1)}_{r,c,h,w} - \sum_{r',h',w'}\delta^{(l+1)}_{r',c,h',w'} \\
&- (X^{(l)}_{r,c,h,w}-\mu_c)(\sigma^2_c+\epsilon)^{-\frac{3}{2}}\sum_{r',h',w'}\delta^{(l+1)}_{r',c,h',w'}(X^{(l)}_{r',c,h',w'}-\mu_{c'})\Bigg)
\end{aligned}
\tag{65}
$$

Derivatan av vikterna hittas på ett liknande sätt med en faktor $\frac{1}{R}$ på grund av att det är medelvärdet för alla exemepl i mini-hopen som är eftersökt. [12] [14]

$$
\begin{aligned}
\frac{\partial L(W)}{\partial \gamma^{(l)}_c} &= \frac{1}{R}\sum_{r}^{R-1}\sum_{c'}^{C'-1}\sum_{h'}^{H'-1}\sum_{w'}^{W'-1}\frac{\partial L(W)}{\partial X^{(l+1)}_{r,c',h',w'}}\frac{\partial X^{(l+1)}_{r,c',h',w'}}{\partial \gamma^{(l)}_c} \\
&= \frac{1}{R}\sum_{r}^{R-1}\sum_{c'}^{C'-1}\sum_{h'}^{H'-1}\sum_{w'}^{W'-1}\delta^{(l+1)}_{r,c',h',w'}\frac{\partial(\gamma^{(l)}_{c'}\hat{X}_{r,c',h',w'}+\beta^{(l)}_{c'})}{\partial \gamma^{(l)}_c} \\
&= \frac{1}{R}\sum_{r}^{R-1}\sum_{c'}^{C'-1}\sum_{h'}^{H'-1}\sum_{w'}^{W'-1}\delta^{(l+1)}_{r,c',h',w'}\hat{X}_{r,c,h',w'}I_{c',c} \\
&= \frac{1}{R}\sum_{r}^{R-1}\sum_{h'}^{H'-1}\sum_{w'}^{W'-1}\delta^{(l+1)}_{r,c,h',w'}\hat{X}_{r,c,h',w'}
\end{aligned}
\tag{66}
$$

$$
\begin{aligned}
\frac{\partial L(W)}{\partial \beta_c^{(l)}} &= \frac{1}{R}\sum_{r}^{R-1}\sum_{c'}^{C'-1}\sum_{h'}^{H'-1}\sum_{w'}^{W'-1}\frac{\partial L(W)}{\partial X_{r,c',h',w'}^{(l+1)}}\frac{\partial X_{r,c',h',w'}^{(l+1)}}{\partial \beta_c^{(l)}} \\
&= \frac{1}{R}\sum_{r}^{R-1}\sum_{c'}^{C'-1}\sum_{h'}^{H'-1}\sum_{w'}^{W'-1}\delta_{r,c',h',w'}^{(l+1)}\frac{\partial(\gamma_{c'}^{(l)}\hat{X}_{r,c',h',w'}+\beta_{c'}^{(l)})}{\partial \beta_c^{(l)}} \\
&= \frac{1}{R}\sum_{r}^{R-1}\sum_{c'}^{C'-1}\sum_{h'}^{H'-1}\sum_{w'}^{W'-1}\delta_{r,c,h',w'}^{(l+1)} I_{c',c} \\
&= \frac{1}{R}\sum_{r}^{R-1}\sum_{h'}^{H'-1}\sum_{w'}^{W'-1}\delta_{r,c,h',w'}^{(l+1)}
\end{aligned}
\tag{67}
$$

4.3 Praktiska Tillämpningar

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

4.3.1 Klassificering av handskrivna siffror

En enkel CNN-modell kan användas för att klassificera handskriva siffror. För att uppnå detta har MNIST DATASET för handskriva siffror används. Den består av 60 000 unika exempel för träning och ytterligare 10 000 exempel för validering av modellen. Valideringsdatan måste vara separerad från datan man tränar med för annars finns det risk att modellen OVERFITTAS och inte kan generalisera för andra exempel än träningsdatan. [17]

Låt modellens prognos betäcknas 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]

Input för modellen är en 28×28 pixelarray. 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 siffra. Om det näst sista lagret är x med samma dimensioner som $\hat{y}$ definieras softmaxfunktionen P genom: [1] [2] [16]

$$
\begin{aligned}
P(x_{r,c}) &= \hat{y}_{r,c} \\
&= \frac{e^{x_{r,c}}}{\sum_{c'=0}^{C-1} e^{x_{r,c'}}}
\end{aligned}
\tag{68}
$$

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

$$\begin{aligned}\frac{\partial P(x_{r,c})}{\partial x_{r,c''}} &= \frac{(\sum_{c'=0}^{C-1} e^{x_{r,c'}})(e^{x_{r,c}})I_{c,c''} - (e^{x_{r,c}})(\sum_{c'=0}^{C-1} e^{x_{r,c'}} I_{c',c''})}{(\sum_{c'=0}^{C-1} e^{x_{r,c'}})^2} \\ &= \frac{(e^{x_{r,c}})I_{c,c''}}{\sum_{c'=0}^{C-1} e^{x_{r,c'}}} - \frac{(e^{x_{r,c}})(e^{x_{r,c''}})}{(\sum_{c'=0}^{C-1} e^{x_{r,c'}})^2} \\ &= \hat{y}_{r,c}(I_{c,c''} - \hat{y}_{r,c''})\end{aligned} \tag{69}$$

Där I är kronecker-deltat.

Tillsammans med softmax används kostnadsfunktionen *cross entropy* betäcknat med L där y är THE GROUND TRUTH: [1] [2]

$$L(W) = -\frac{1}{R}\sum_{r=0}^{R-1}\sum_{c=0}^{C-1} y_{r,c}\ \log \hat{y}_{r,c} \tag{70}$$

$$\begin{aligned}\frac{\partial L(W)}{\partial \hat{y}_{r',c'}} &= \frac{\partial\left(-\frac{1}{R}\sum_{r=0}^{R-1}\sum_{c=0}^{C-1} y_{r,c}\ \log \hat{y}_{r,c}\right)}{\partial \hat{y}_{r',c'}} \\ &= -\frac{1}{R}\sum_{r=0}^{R-1}\sum_{c=0}^{C-1} y_{r,c}\frac{\partial\left(\log \hat{y}_{r,c}\right)}{\partial \hat{y}_{r',c'}} \\ &= -\frac{1}{R}\sum_{r=0}^{R-1}\sum_{c=0}^{C-1} y_{r,c}\frac{1}{\hat{y}_{r,c}} I_{r,r'} I_{c,c'} \\ &= -\frac{1}{R}\frac{y_{r',c'}}{\hat{y}_{r',c'}}\end{aligned} \tag{71}$$

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.

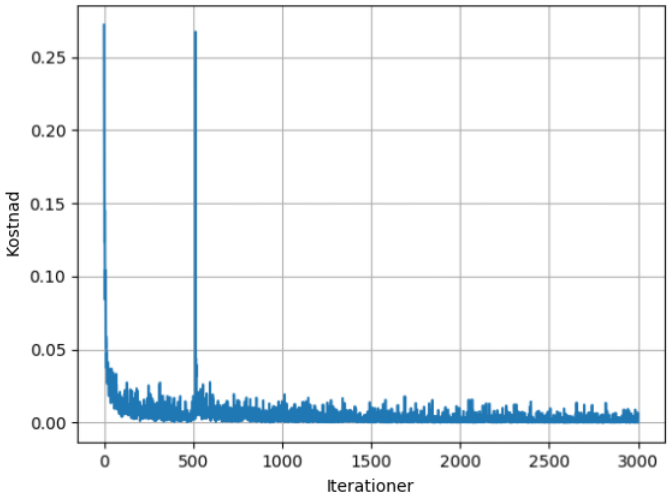
Modell 1
Input
BN
conv3 c16 s1
maxpool2 s2
ReLU
BN
conv3 c16 s1
ReLU
BN
conv4 c16 s1
maxpool2 s2
ReLU
BN
conv4 c10 s1
softmax
Output

Modell 2
Input
BN
conv3 c16 s1
ReLU
BN
conv3 c16 s1
ReLU
BN
conv3 c16 s1
maxpool2 s2
ReLU
BN
conv4 c16 s1
ReLU
BN
conv4 c24 s1
ReLU
BN
conv3 c24 s1
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 presternade nätverket, modell 2, på fler iterationer tills validationsprecisionen började avta. Den testades varje 60 000 exemepel eller efter varje 600 iterationer med en hopstorlek av 100. Efter varje 60 000 bilder blandades dem för att förbättra inlärningen.

Den slutgiltiga precisionen blev 99.2%. Av 10 000 handskrivna siffror lyckades



Figur 10: En graf av kostnadsfunktionen efter varje iteration av mini-hopen av 100 bilder.

modellen klassifisera 9 919 siffror rätt.

4.3.2 Objektdetektering i bilder

Om modellen YOLO:

4.3.3 Ansiktsigenkänning

anpassar yolo till en ansiktsdatabas

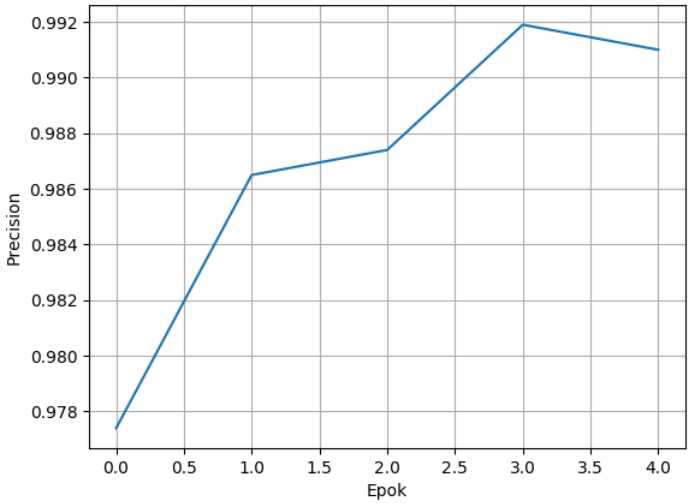
5 Diskussion

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. 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.



Figur 11: En graf av validationsprecisionen efter varje epok av 60 000 bilder med modell 2.

[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.* Standford 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.