Konvolutionella neurala nätverk för ansiktsigenkänning och sifferavläsning

27 februari 2018
Nikita Zozoulenko Na15b
nikita.zozoulenko@gmail.com
Katedralskolan i Linköping
Handledare: Rickard Engström
VT 2017 - HT 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 cutting edge face detector, successfully detecting over 50 faces in a crowded scene.

Innehåll

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 människans hjärna är uppbyggd av ett flertal neuroner som är kopplade till varandra. Yann LeCunn uppfanna det Konvolutionella Neurala Nätverket 1998 och det var designat till att efterlikna hur ett djurs prefrontalkortex fungerar. Trots att neurala nätverk har funnits i mer än ett halvt sekel har dem inte sett stor använding innan år 2012 då 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 olika språk och automatiskt skapande av undertexter från ljud. [1]

1.2 Syfte

Syftet med arbetet är att härleda och redogöra för den underliggande matematiken bakom den matematiska modellen av artificiella neurala nätverk och 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

Majoriteten av tiden gick åt till se på videoföreläsningar av kursen "CS231n: Convolutional Neural Networks for Visual Recognitionfrån 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 beräknas i modellen jämfördes med en analytitisk approximation med hjälp av derivatans definition. Flera forskares bloggar användes för att i mer detalj förstå matematiken bakom nätverksoperationerna. För att fördjupa mig i konvolutionella neurala nätverk läste jag de riktiga vetenskapliga artiklarna såsom Batch Normalization (Ioffe & Szegedy, 2015) och Focal Loss (Lin et al. 2017) för att sedan implementera dem i python. När min egen implementation blev för beräkningsineffektiv började jag använda två GPU-accelererande maskininlärningsbibliotek: PyTorch och Tensorflow.

3 Resultat

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

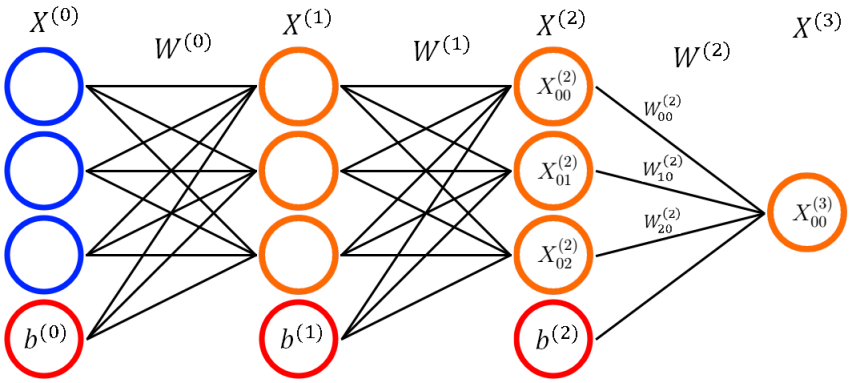
Ett artificellt neuralt nätverk består av ett antal lager neuroner. De är uppbyggda rekursivt så att resultatet av ett lager är inmatningsdatan till nästintilliggande lager. Inmatningsdatan för modellen matas in i det första lagret sådant att ett neuron är ett värde. Nervsignalen propageras framåt tills 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 *Densly 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 träningsexempel bearbetas samtidigt i en så kallad mini-hop med storlek R. Låt $X_{ri}^{(l)}$ benämna neuron nummer i i lager l i träningsexempel 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 $X^{(l)}_{ra}$ och $X^{(l+1)}_{rb}$. 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]

$$Z^{(3)}_{r0} = X^{(2)}_{r0}W^{(2)}_{r0} + X^{(2)}_{r1}W^{(2)}_{10} + X^{(2)}_{r2}W^{(2)}_{20} + b^{(2)} \tag{1}$$

$$X^{(3)}_{r0} = f(Z^{(3)}_{r0}) \tag{2}$$

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

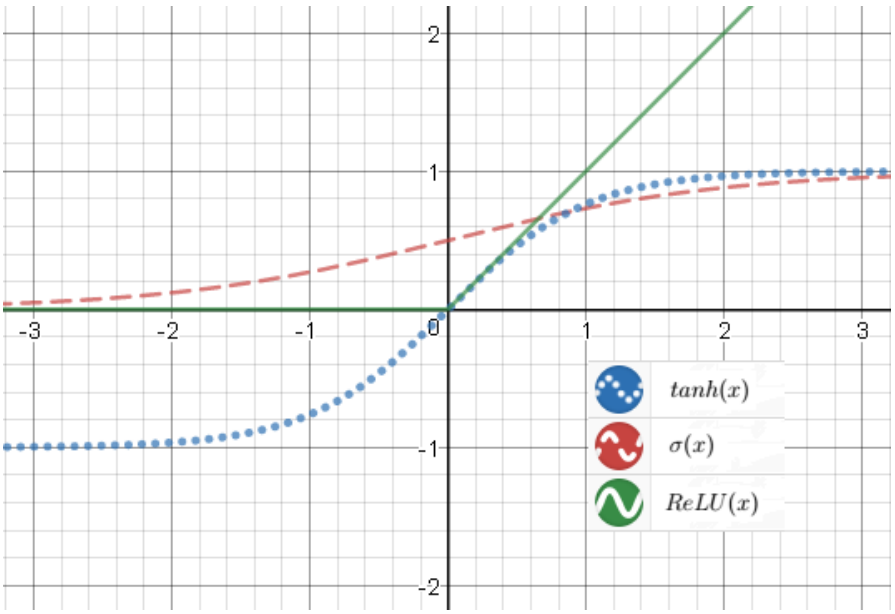
Neuronmatriserna och viktmatriserna är definierade på ett sådant sätt att framåtpropegationen mellan lager l och $l+1$ kan beräknas med en matrismultiplikation och kan därför beräknas rekursivt från inputneuronerna: [1] [6]

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

$$X^{(l+1)}_{r,i} = f(Z^{(l+1)}_{ri}) \tag{4}$$

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 bakåtpropagering. 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 $tanh$.

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

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

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

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]

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

Träningen av det neurala nätverket är processen där man hittar konstnadsfunktionens 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 $W^{(0)}$, $W^{(1)}$, ..., $W^{(l)}$, $b^{(0)}$, $b^{(1)}$, ..., $b^{(l)}$ som definieras genom: [7] [3]

$$\nabla L(\theta) : \mathbb{R}^n \to \mathbb{R}^n \tag{9}$$

$$\nabla L(\theta) = \begin{pmatrix} \frac{\partial L(\theta)}{\partial \theta^{(0)}} & \frac{\partial L(\theta)}{\partial \theta^{(1)}} & \dots & \frac{\partial L(\theta)}{\partial \theta^{(n-1)}} \end{pmatrix} \tag{10}$$

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

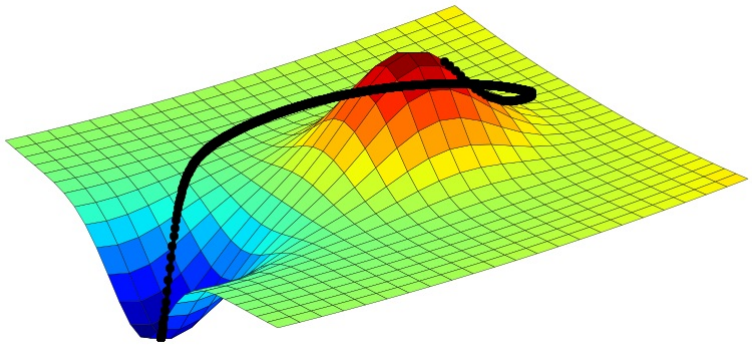
$$\frac{\partial L(\theta)}{\partial \theta^{(l)}} = \nabla_{\theta^{(l)}} L(\theta) \tag{11}$$

$$\theta^{(l)} \to \theta^{(l)} - \alpha \frac{\partial L(\theta)}{\partial \theta^{(l)}} \tag{12}$$

Nätverket tränas genom att varje träningsexempel (X, y) framåtpropageras och insätts i kostnadsfunktionen 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.[17]

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

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

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

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

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

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

$\delta^{(l)}$, 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 användingen 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 ut slumpvist. En iteration av SGD appliceras genom att framåtpropagera ett träningsexempel och sedan bakåtpropagera felet för att beräkna gradienten av kostnadsfunktionen med avseende på alla vikter i närverket. 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`

3.4 Konvolutionella neuala 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 horizontell kantdetektering har sammanrullat över en bild av en katt.

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 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]

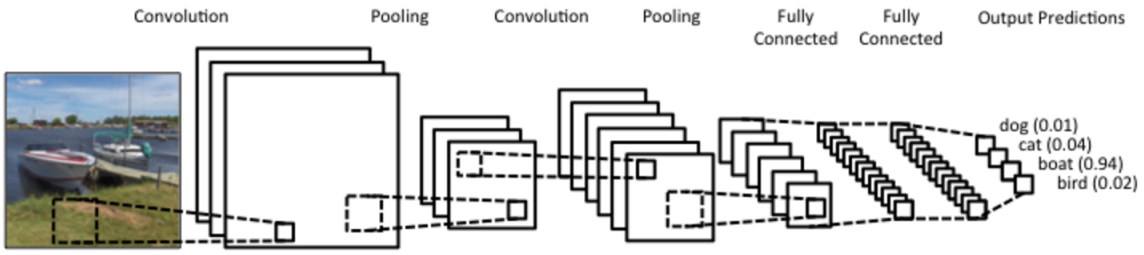
$$X^{(0)} \xrightarrow{W^{(l)}} X^{(1)} \xrightarrow{W^{(l)}} \dots \xrightarrow{W^{(l_{sista}-2)}} X^{(l_{sista}-1)} \xrightarrow{W^{(l_{sista}-1)}} X^{(l_{sista})} = \hat{y} \quad (19)$$

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 paritella 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]

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]

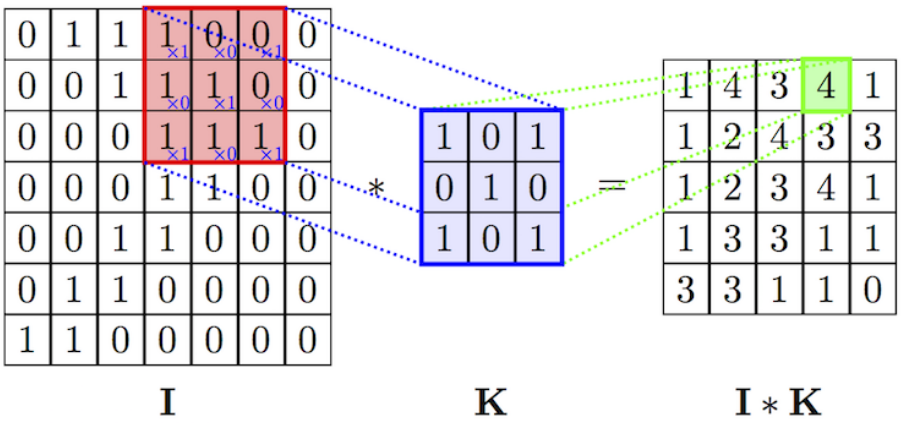


Figur 5: En illustration av ett konvolutionellt neuralt nätverk. Varje skiva är en egen *feature map*. [18]

3.4.1 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. Konvolutionsoperatorn betecknas med $*$. [1] [3]

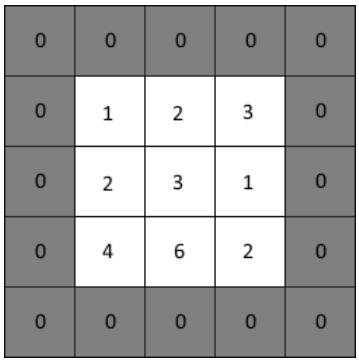


Figur 6: En kärna med storlek 3×3 sammanrullar över ett område med dimensioner 6×6 och bildar en aktivering med dimensionerna 4×4. [19]

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]



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]

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

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

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

$$w = sw' \tag{22}$$

$$h = sh' \tag{23}$$

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

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 möjliggör att en konvolution att beräknas med en matrismultiplikation. [1] [3] [4]

3.4.2 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 $\frac{\partial L(W)}{\partial X^{(l+1)}_{r,c',h',w'}}$ 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 likt ekvationer (15), (16) och (18). $X^{(l+1)}_{r,c',h',w'}$ byts sedan ut mot dess definition enligt ekvation (24). [3] [10] [12] [14]

$$\begin{aligned}
\delta^{(l)}_{r,c,h,w} &= \frac{\partial L(W)}{\partial X^{(l)}_{r,c,h,w}} \\
&= \sum_{c'=0}^{C'-1}\sum_{h'=0}^{H'-1}\sum_{w'=0}^{W'-1} \frac{\partial L(W)}{\partial X^{(l+1)}_{r,c',h',w'}} \frac{\partial X^{(l+1)}_{r,c',h',w'}}{\partial X^{(l)}_{r,c,h,w}} \\
&= \sum_{c'=0}^{C'-1}\sum_{h'=0}^{H'-1}\sum_{w'=0}^{W'-1} \delta^{(l+1)}_{r,c',h',w'} \frac{\partial \sum_{c=0}^{C-1}\sum_{j=0}^{k_H-1}\sum_{i=0}^{k_W-1} X^{(l)}_{r,c,h'+j,w'+i} W^{(l+1)}_{c',c,j,i}}{\partial X^{(l)}_{r,c,h,w}}
\end{aligned} \tag{25}$$

Varje produkt i den innersta summan kommer att vara lika 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: [10] [12] [14]

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

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 $rot()$. [10] [12] [14]

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

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: [1] [10] [12] [14]

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

3.4.3 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]

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

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

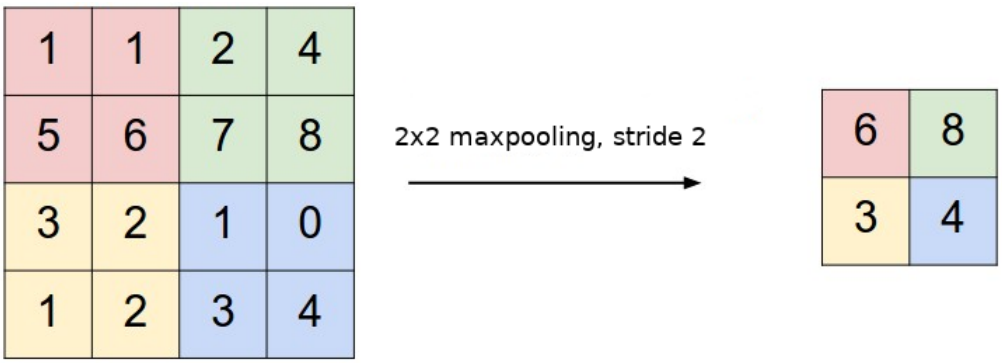
3.4.4 Aktiveringsfunktionslager bakåtpropagering

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]

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

3.4.5 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 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 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]

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

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

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

Antal kanaler förblir konstant. [1] [3]

Matematiskt beskrivs maxpoollagret genom: [1] [3]

$$X^{(l+1)}_{r,c',h',w'} = \max_{0 \le j < k,\ 0 \le i < k} X^{(l)}_{r,c',(h's+j),(w's+i)} \tag{34}$$

3.4.6 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 (34): [1] [3] [14]

$$\begin{aligned}
\delta^{(l)}_{r,c,h,w} &= \frac{\partial L(W)}{\partial X^{(l)}_{r,c,h,w}} \\
&= \frac{\partial L(W)}{\partial X^{(l+1)}_{r,c',h',w'}} \frac{\partial X^{(l+1)}_{r,c',h',w'}}{\partial X^{(l)}_{r,c,h,w}} \\
&= \delta_{r,c',h',w'} \frac{\partial \max\limits_{0 \le j < k, 0 \le i < k} X^{(l)}_{r,c',(h's+j),(w's+i)}}{\partial X^{(l)}_{r,c,h,w}}
\end{aligned} \tag{35}$$

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] [14]

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

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] [14]

3.4.7 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. Följaktligenhar bearbetningen av flera exemepl i taget i en mini-hop en annan praktiska tillämpning än att enbart beräkna konvolutionerna mer tidseffektivt. [1] [9]

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] [9]

Först beräknas medelvärdena μ_c och varianserna σ_c^2 till varje *feature map* c: [1] [9]

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

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

Sedan beräknas den normaliserade aktiveringen $\hat{X}$. Epsilon används för numerisk stabilitet. [1] [9]

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

Sist introduceras 2 vikter, $\gamma^{(l)}_{c'}$ och $\beta^{(l)}_{c'}$, vilka tillåter nätverket att upphäva normaliseringen om nätverket dömmer det att vara användbart. [1] [9]

$$X^{(l+1)}_{r,c,h,w} = \gamma^{(l)}_c \hat{X}_{r,c,h,w} + \beta^{(l)}_c \tag{40}$$

När nätverket ska köras utanför träning, också kallat *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 databaser, 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] [9]

$$\mu_{EWMA_c} \to \lambda\mu_c + (1-\lambda)\mu_{EWMA_c} \tag{41}$$

$$\sigma^2_{EWMA_c} \to \lambda\sigma_c^2 + (1-\lambda)\sigma^2_{EWMA_c} \tag{42}$$

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

3.4.8 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 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: [11] [13]

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

$$\frac{\partial a_j}{\partial a_i} = I_{i,j} \tag{44}$$

$$\sum_j a_i I_{i,j} = a_j \tag{45}$$

Först bryts $\frac{\partial L(W)}{\partial X^{(l)}}$ upp i tre bråk och sedan summeras alla partiella derivator likt ekvation (25). [11] [13] Här summeras dessutom R-dimensionen på grund av att aktiveringar från hela mini-hopen har en påverkan på $\delta^{(l)}$.

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

$\frac{\partial L(W)}{\partial X^{(l+1)}_{r,c,h,w}}$ 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. [11] [13]

$$\begin{aligned} \frac{\partial X^{(l+1)}_{r',c',h',w'}}{\partial \hat{X}_{r',c',h',w'}} &= \frac{\partial(\gamma^{(l)}_{c'}\hat{X}_{r',c',h',w'} + \beta^{(l)}_{c'})}{\partial \hat{X}_{r',c',h',w'}} \\ &= \gamma^{(l)}_{c'} \end{aligned} \tag{47}$$

För derivatan av den centrerade aktiveringen med avseende på den originella aktiveringen tillämpas produktregeln: [11] [13]

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

Derivatan av den första faktorn med avseende på aktiveringen beräknas med hjälp av ekvationer (43), (44) och (45). [11] [13]

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

Derivatan av den andra faktorn med avseende på aktiveringen beräknas på ett liknande sätt med hjälp av kedjeregeln och ekvationer (43), (44) och (45). [11] [13]

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

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:

$$
\begin{aligned}
\delta^{(l)}_{r,c,h,w} &= \sum_{r'=0}^{R-1}\sum_{c'=0}^{C'-1}\sum_{h'=0}^{H'-1}\sum_{w'=0}^{W'-1}\frac{\partial L(W)}{\partial X^{(l+1)}_{r',c',h',w'}}\frac{\partial X^{(l+1)}_{r',c',h',w'}}{\partial \hat{X}_{r',c',h',w'}}\frac{\partial \hat{X}_{r',c',h',w'}}{\partial X^{(l)}_{r,c,h,w}} \\
&= \sum_{r',c',h',w'}\delta^{(l+1)}_{r',c',h',w'}\gamma^{(l)}_{c'}(\sigma^2_{c'})^{-\frac{1}{2}}(I_{r',r}I_{c',c}I_{h',h}I_{w',w} - \frac{1}{RHW}I_{c',c}) \\
&\quad - \sum_{r',c',h',w'}\delta^{(l+1)}_{r',c',h',w'}\gamma^{(l)}_{c'}\frac{1}{RHW}(X^{(l)}_{r',c',h',w'} - \mu_{c'})(X^{(l)}_{r,c',h,w} - \mu_{c'})(\sigma^2_{c'})^{-\frac{3}{2}}I_{c',c} \\
&= \delta^{(l+1)}_{r,c,h,w}\gamma^{(l)}_{c}(\sigma^2_{c})^{-\frac{1}{2}} - \frac{1}{RHW}\sum_{r',h',w'}\delta^{(l+1)}_{r',c,h',w'}\gamma^{(l)}_{c}(\sigma^2_{c})^{-\frac{1}{2}} \\
&\quad - \frac{1}{RHW}\sum_{r',h',w'}\delta^{(l+1)}_{r',c,h',w'}\gamma^{(l)}_{c}(X^{(l)}_{r',c,h',w'} - \mu_{c'})(X^{(l)}_{r,c,h,w} - \mu_{c})(\sigma^2_{c})^{-\frac{3}{2}} \\
&= \frac{1}{RHW}\gamma^{(l)}_{c}(\sigma^2_{c})^{-\frac{1}{2}}\Bigg(RHW\delta^{(l+1)}_{r,c,h,w} - \sum_{r',h',w'}\delta^{(l+1)}_{r',c,h',w'} \\
&\quad - (X^{(l)}_{r,c,h,w} - \mu_{c})(\sigma^2_{c})^{-\frac{3}{2}}\sum_{r',h',w'}\delta^{(l+1)}_{r',c,h',w'}(X^{(l)}_{r',c,h',w'} - \mu_{c'})\Bigg)
\end{aligned}
\tag{51}
$$

Derivatan av vikterna hittas på ett liknande sätt som ekvation (46) till (51). [11] [13]

$$
\begin{aligned}
\frac{\partial L(W)}{\partial \gamma^{(l)}_{c}} &= \sum_{r}^{R-1}\sum_{c'}^{C'-1}\sum_{h'}^{H'-1}\sum_{w'}^{W'-1}\frac{\partial L(W)}{\partial X^{(l+1)}_{r,c',h',w'}}\frac{\partial X^{(l+1)}_{r,c',h',w'}}{\partial \gamma^{(l)}_{c}} \\
&= \sum_{r}^{R-1}\sum_{c'}^{C'-1}\sum_{h'}^{H'-1}\sum_{w'}^{W'-1}\delta^{(l+1)}_{r,c',h',w'}\frac{\partial(\gamma^{(l)}_{c'}\hat{X}_{r,c',h',w'} + \beta^{(l)}_{c'})}{\partial \gamma^{(l)}_{c}} \\
&= \sum_{r}^{R-1}\sum_{c'}^{C'-1}\sum_{h'}^{H'-1}\sum_{w'}^{W'-1}\delta^{(l+1)}_{r,c',h',w'}\hat{X}_{r,c,h',w'}I_{c',c} \\
&= \sum_{r}^{R-1}\sum_{h'}^{H'-1}\sum_{w'}^{W'-1}\delta^{(l+1)}_{r,c,h',w'}\hat{X}_{r,c,h',w'}
\end{aligned}
\tag{52}
$$

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

3.4.9 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 klassificiera 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:

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

3.4.10 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] [15]

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

3.5 Praktiska Tillämpningar

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

3.5.1 Klassificering av handskrivna siffror

En enkel CNN-modell kan användas för att klassificera handskriva siffror. För att uppnå detta har MNISTdatabasen för handskriva siffror används. Den består av 60 000 unika exempel av handskrivna siffror. [16]

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]

Input för modellen är en $R \times 1 \times 28 \times 28$ tensor 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 siffra. Konstandsfunktionen som har minimerats under träningstiden är funktionen *cross-entropy* betecknat med L. Den verkar på två sannolikhetsfördelningar: De verkliga sannolikheterna

Figur 9: Tio bilder av handskrivna siffror från MNISTdatabasen. [16]

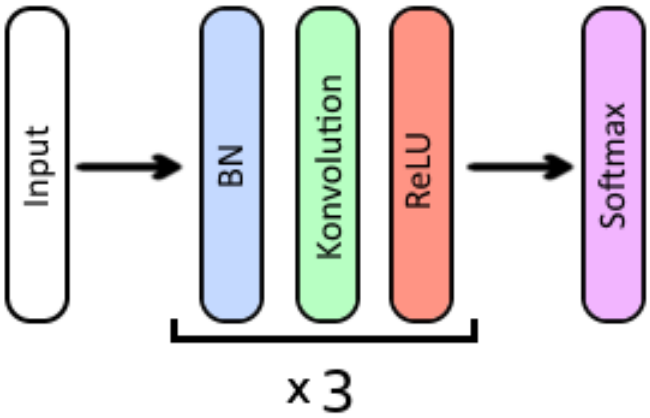
y och de prognostiserade sannolikheterna $\hat{y}$: [1] [2]

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

$$\frac{\partial L(W)}{\partial \hat{y}_{r',c'}} = -\frac{y_{r',c'}}{\hat{y}_{r',c'}} \tag{57}$$

Endast en modell tränades på grund av begränsningar i datakraft. Den bestod av tre stycken konvolutionsblock, följt med ett softmaxlager (se figur 10). En mini-hopstorlek av 50 användes och 5000 iterationer av framåt- och bakåtpropagering kördes på min egenimplementerade modell skriven i python. Totalt tog det 4 timmar att träna nätverket på min CPU.

Den slutgiltiga precisionen blev 99.2% på 10 000 nya handskrivna siffror modellen aldrig hade sett tidigare. Av 10 000 handskrivna siffror lyckades modellen klassifisera 9 919 siffror rätt.



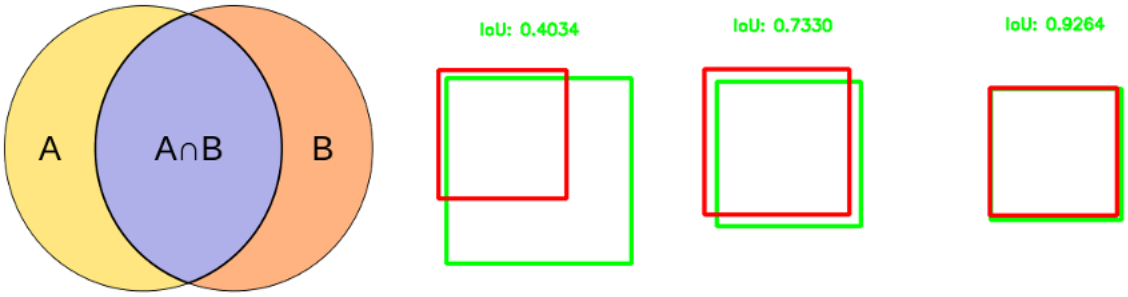
Figur 10: Modellen som användes för att läsa av handskrivna siffror. Batch Normalization benämns med BN, konvolution menas en konvolution med kärnstorlek k = 5 och stride s = 2. ReLU är aktiveringsfunktionen Rectified Linear Units och softmax är softmaxlagret.

3.5.2 Ansiktsigenkänning

Ett konvolutionellt neuralt nätverk är anpassat för att detektera upp till flera tusen ansikten åt gången för realtidsvideo. Modellen producerar ett bestämt antal *boudning boxes*: koordinater som ska detektera alla olika objekt i bilden. Jag utgick från architekturen från RetinaNet från *Focal Loss for Dense Object Detection* (Lin et al.) och anpassade den till $K = 1$ klasser. I varje position i outputlagret föreslår modellen en *boudning box* med fyra koordinater: två punkter för det övre vänstra hörnet respektive nedre hörna hörnet av den positionens *bounding box*. Utöver det förutsägs det $K + 1$ sannolikheter att *bounding boxen* innehåller ett objekt av alla K förgrundklasser och 1 bakgrundklass (lådan innehåller inga objekt).[1][23]

Modellen för objektdetektering går ut på att man utgår från ett antal så kallade anchor boxes vid varje rumslig position i det sista konvolutionella lagret. Om lagret har bredden och höjden W respektive H och A olika storlekar på anchor boxes har lagret totalt WHA olika anchor boxes. Vid träning tilldelas en anchor box ett objekt om dens *intersection over union* (IoU) med den verkliga bounding boxen är större än 0.5 (se figur 11 IoU används för att beräkna hur lika två olika mängder är. I detta fall är mängderna areorna av en anchor box och ett objekts verkliga bounding box. IoU definieras som storleken av snittet av två mänger A och B dividerat med storleken av unionen av A och B: [1] [22]

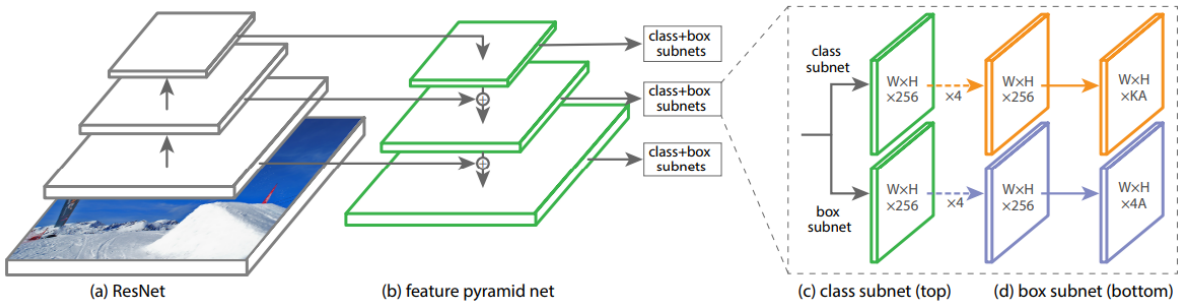
$$IoU(A, B) = \frac{|A \cap B|}{|A \cup B|} \tag{58}$$



Figur 11: IoU definieras som storleken av snittet av två mänger A och B dividerat med storleken av unionen av A och B. En större IoU medför att prognosen är närmare sanningen. [22]

RetinaNet använder sig av en *feature pyramid*-arkitektur beskriven i *Feature Pyramid Networks for Object Detection"* (Lin et al.) och en ResNet101 (He et al.) som *backbone*. Featurepyramiden används för att få ut *feature maps* från olika delar av det konvolutionella neurala nätverket för att kunna detektera

objekt av olika storlekar. Varje pyramidnivå matas in i ett klassifikationshuvud respektive regressionshuvud (se figur 12) för att räkna ut sannolikheterna att varje anchor box innehåller de $K + 1$ olika klasserna, samt att förfina varje anchor box koordinater genom att prognostisera 4 olika offsets för varje sida av anchor boxens bounding box. [21] [23] [24]



Figur 12: RetinaNets arkitektur från *Focal Loss for Dense Object Detection* (Lin et al.). Varje pyramidnivå matas in i ett klassifikationshuvud och ett regressionshuvud. [23]

Till skillnad från RetinaNet har min modell, FaceNet, ett mindre antal pyramidnivåer för att lätta på beräkningskraften och för att detektera mindre objekt. RetinaNets anchors har areor från 32^2 till 512^2 pixlar spridda över pyramidnivåer P3 till P7. FaceNet använder sig av areor 14^2 till 220^2 pixlar spridda över pyramidnivåer P3 till P6. Regressionshuvudet och klassifikationshuvudet i RetinaNet består av 5 konvolutionslager med kärnstorlek 3. Facenet använder istället 2 och 3 konvolutionslager för regressionshuvudet respektive klassifikationshuvudet. I det originala Feature Pyramidnätverket användes linjär interpolation för att göra storleken av högre feature maps större. I FaceNet används k-nearest neighbors (k-NN) istället. ResNet50 användes istället för ett ResNet101 som backbone för att lätta på beräkningskostnaderna. [21] [23] [24]

Kostnadsfunktionen som användes var en kombination av Focal Loss (Lin et al.) och Smooth L1 Loss. FaceNet använder samma hyperparametrar som uppnådde bästa resultat för RetinaNet: $\alpha = 3$ för förgrundklassen och $\gamma = 2$. [23]

$$L_r(x) = \begin{cases} 0.5x^2 & \text{om } |x| < 1 \\ |x| - 0.5 & \text{i annat fall} \end{cases} \tag{59}$$

$$L_c(p, \hat{p}) = -\alpha(1-p)^\gamma p \log \hat{p} \tag{60}$$

$$L(W) = \sum_{k \in pyramid} \frac{1}{N_c^k} \sum_{a \in anchors} L_r(r_a - \hat{r}_a) + \sum_{k \in pyramid} \frac{1}{N_c^r} \sum_{a \in anchors} L_c(p_a, \hat{p}_a) \tag{61}$$

N_c^k och N_r^k betäcknar alla anchors som deltar i klassifikationskonstnaden respektive regressionskonstnaden i pyramidnivå k. För klassifikationskonstnaden är $\hat{p}$ 1 för alla anchors som har blivit tillgivna ett objekt och 0 för alla som bara innehåller bakgrundsklassen. Endast anchors som innehåller ett objekt deltar i regressionskostnaden. r_a innehåller de 4 koordinaterna den slutgiltiga boudning boxen har.

Modellen tränades i 350 000 iterationer med en mini-hopsstorlek av 3. Träningshastigheten började på 0.005 och dividerades med 10 vid 200 000 iterationer. Varje träningsexempels storlek gjordes slumpmässigt om till 512^2, 576^2 eller 640^2 pixlar och speglades horizontellt med sannolikhet 0,5 för att artificiellt utöka mängden träningsdata. Den totala träningstiden var 18 timmar på en NVIDIA GTX 1080ti. Figur 13 visar kvantitativa resultat från WIDERFace valideringsdata.

En iteration av framåtpropageringen tar 20 ms vilket möjliggör att modellen kan köras i realtid, 50 gånger i sekunden.

4 Diskussion

Den matematiska modellen av artificiella neurala nätverk och konvolutionella neurala nätverk, samt bakåt- och framåtpropagering förklaras och härleds i sektion 3.2 och 3.3. Ett exempel på hur modellen kan tillämpas är att klassificera handskrivna siffror, som i sektion 3.4.1. En annan möjlighet är att anpassa ett konvolutionellt neuralt nätverk för att detektera alla ansikten i en digital bild. På grund av att en iteration av algoritmen tar 20 ms är det möjligt att köra modellen i realtid. Det kan exempelvis användas i övervakningskameror eller inom robotik för att känna igen människoansikten i realtid.

Algoritmen är dock inte perfekt och har flera brister. Modellen prognostiserar att det finns ansikten på flera ställen där det inte finns några i verkligheten. För att undankomma detta låter jag endast modellen visa upp bounding boxes med sannolikhet större än 0.5 att det finns ett ansikte där. Resultatet

Figur 13: Kvantitativa resultat från valideringsdatan från WIDERFace: Bilder som modellen aldrig har sett tidigare.

blir att modellen inte hittar 100% av all ansikten i bilden (se figur 13). Ytterligare har modellen problem med små ansikten. Detta är dels på grund av att modellen inte kan urskilja ett ansikte med så lite information (ca minimum 10^2 pixlar per ansikte), och dels för att FaceNet prognostiserar ca 30 000 olika anchors för de minsta storlekarna. Detta leder till att klassifikationskostnaden består av till mestadels negativa exempel. Delta-felet av bakgrundsklasserna dominerar och gör att delta-felet av förgrundsklassen (ansikten) inte har en stor påverkan på den slutgiltiga gradienten.

5 Källkritik

5.1 CS231n: Convolutional Neural Networks for Visual Recognition

Angående äkthetskriteriet är källan en kurs utgiven av Stanford, ett av världens mest prestigefulla universitet. Kursen är gjord av Fei-Fei Li och Andrej Karpathy vilka är experter inom området Deep learning. Fei-Fei Li har varit författare för över 100 olika publicerade vetenskapliga artiklar inom maskininlärning och är professor på Stanford. Andrej Karpathy är en doktorand från Stanford och har arbetet för bland annat Google, DeepMind och OpenAI. Just nu arbetar han som Director of AI"hos Tesla. Informationen i kursen kommer från experter inom området.

Enligt tendenskriteriet är källan opartisk. Stanford är en av de mest prestigefulla universiteten i världen och skulle inte riskera att ge ut felaktig eller vinklad information. Det skulle dessutom försämra föreläsarnas rykten inom forskarvärlden.

Angående beroendekriteriet är källan en kombination av att vara en förstahandskälla och andrahandskälla. Å ena sidan kan Li och Karpathy bidra med deras egen forskning som en förstahandskälla, men å andra sidan är majoriteten av undervisningsmaterialet inte något som förelasarna själv har kommit på.

Enligt tidskriteriet ligger källan bra till. Kursen är endast två år gammal och innehåller den nyaste tekniken inom deep learning. Trots att maskininlärning är ett snabbt växande område spelar det inte en stor roll att kursen är två år gammal för att kursen lär ut grunderna inom konvolutionella neurala nätverk för dataseende: De nya upptäckterna som sker efter kursen gavs ut är inte relevanta för målgruppen.

5.2 Batch normalization: Accelerating deep network training by reducing internal covariate shift.

Källan är en vetenskaplig artikel av Sergey Ioffe och Christian Szegedy. De är båda doktorander och forskare på Google. Deras forsking och denna källa har varit kritisk för den fortsatta utvecklingen för neurala nätverk och deep learning. Enligt äkthetskriteriet är källan trovärdig. Pappret har genomgått peer review och har implementerats hundratals gånger av olika företag och privatpersoner i flera olika modeller. Detsamma gäller tendenskriteriet.

Enligt beroendekriteriet är källan en förstahandskälla. Källan beskriver forskningen som Ioffe och Szegedy har genomfört.

Angående tidskriteriet är källan av hög kvalité. Den vetenskapliga artikeln var publicerad år 2015 och innehåller moderna tekniska framsteg inom deep learning. Källan är digitaliserad och har ingen risk att bli utsatt för narrativ smitta på grund av att originalpappret är spriden och arkiverad över nätet.

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.* 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] *Batch normalization: Accelerating deep network training by reducing internal covariate shift.* S. Ioffe och C. Szegedy. arXiv preprint arXiv:1502.03167, 2015.

[10] *Backpropagation In Convolutional Neural Networks.* J. Kafunah. DeepGrid, Organic Deep Learning. URL http://www.jefkine.com/. 5 september 2016.

[11] *What does the gradient flowing through batch normalization looks like?* C. Thorey. Machine Learning Blog. URL http://cthorey.github.io/. 28 januari 2016.

[12] *Note on the implementation of a convolutional neural networks.* C. Thorey. Machine Learning Blog. URL http://cthorey.github.io/. 2 februari 2016.

[13] *Understanding the backward pass through Batch Normalization Layer.* Flaire of Machine Learning URL https://kratzert.github.io. 5 september 2016.

[14] *Convolutional Neural Networks.* A. Gibiansky. URL http://andrew.gibiansky.com. 24 februari 2014.

[15] *Classification and Loss Evaluation - Softmax and Cross Entropy Loss.* P. Dahal. DeepNotes. URL https://deepnotes.io/softmax-crossentropy. 24 februari 2014.

[16] *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.

[17] *Pygradsc.* J. Komoroske. URL https://github.com/joshdk/pygradesc. 12 oktober 2012.

[18] *Understanding Convolutional Neural Networks for NLP.* D. Britz. WildML, Artificial Intelligence, Deep Learning, and NLP. 7 november 2015.

[19] *Understanding Convolutional Neural Networks for NLP.* D. Britz. WildML, Artificial Intelligence, Deep Learning, and NLP. 7 november 2015.

[20] *Deep learning for complete beginners: convolutional neural networks with keras.* P. Veličković. Camebridge Spark. URL https://cambridgespark.com/content. Senast uppdaterad 20 mars 2017.

[21] *Deep residual learning for image recognition.* K. He, X. Zhang, S. Ren, och J. Sun. arXiv preprint arXiv:1512.03385, 2015.

[22] Jaccard Index. Wikipedia. URL https://en.wikipedia.org/wiki/Jaccard_index. Hämtad 20 januari 2018

[23] *Focal Loss for Dense Object Detection.* Tsung-Yi Lin, Priya Goyal, Ross B. Girshick, Kaiming He and Piotr Dollár. arXiv preprint arXiv:1708.02002, 2017

[24] *Feature Pyramid Networks for Object Detection.* Tsung-Yi Lin, Piotr Dollár, Ross B. Girshick, Kaiming He, Bharath Hariharan and Serge J. Belongie. arXiv preprint arXiv:1612.03144, 2016

[25] *WIDER FACE: A Face Detection Benchmark.* Yang, Shuo and Luo, Ping and Loy, Chen Change and Tang, Xiaoou IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016