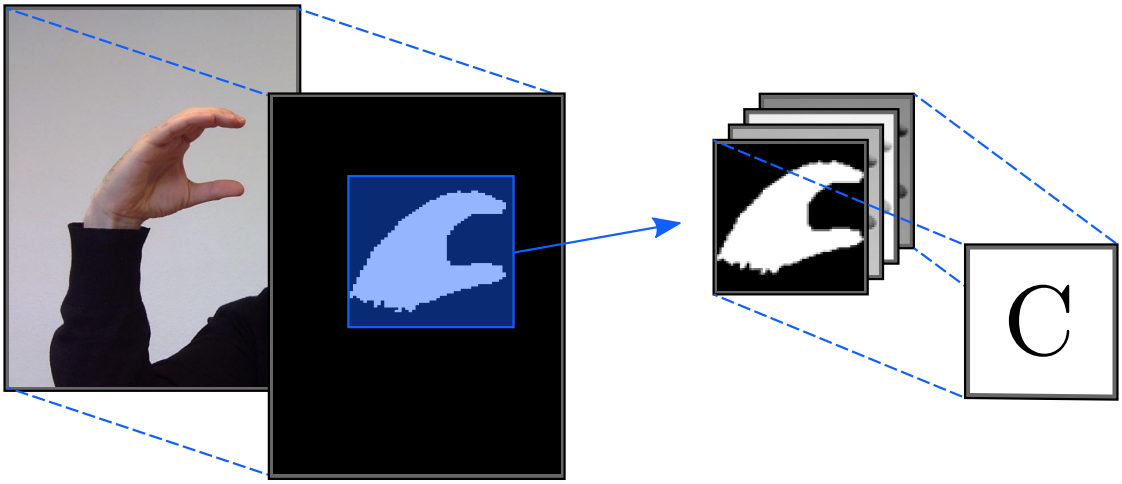
CHALMERS



Automatiserad teckenspråkstolkning genom deep learning-metoder

Tillämpning av neurala faltningsnätverk för datorseende

Kandidatarbete, Teknisk fysik

Institutionen för Signaler och system
CHALMERS TEKNISKA HÖGSKOLA
Göteborg, Sverige 2016

Kandidatarbete SSYX02-16-35

Automatiserad teckenspråkstolkning genom deep learning-metoder

Tillämpning av neurala faltningsnätverk för datorseende

Anton Älgmyr	algmyr@student.chalmers.se
Erik Larsson	erlarsso@student.chalmers.se
Felix Viberg	felixv@student.chalmers.se
Jacob Steffenburg	jacobst@student.chalmers.se
Svante Karlsson	ksvante@student.chalmers.se

CHALMERS

Institutionen för Signaler och system
Chalmers tekniska högskola
Göteborg, Sverige 2016

Automatiserad teckenspråkstolkning genom deep learning-metoder
Tillämpning av neurala faltningsnätverk för datorseende
Anton Älgmyr
Erik Larsson
Felix Viberg
Jacob Steffenburg
Svante Karlsson

© Anton Älgmyr, Erik Larsson, Felix Viberg, Jacob Steffenburg, Svante Karlsson, 2016.

Handledare: Måns Larsson, Signaler och system
Examinator: Fredrik Kahl, Signaler och system

Kandidatarbete SSYX02-16-35
Institutionen för Signaler och system
Chalmers tekniska högskola

Omslag: Diagram som illustrerar den segmentering och klassificering som görs med de neurala nätverk som utvecklats.

Typsatt i LaTeX
Publicerat digitalt via Chalmers Publication Library (CPL)
Göteborg, Sverige 2016

Automatiserad teckenspråkstolkning genom deep learning-metoder
Tillämpning av neurala faltningsnätverk för datorseende
Anton Älgmyr, Erik Larsson, Felix Viberg, Jacob Steffenburg, Svante Karlsson

Institutionen för Signaler och system
Chalmers tekniska högskola

Abstract

The rapid advancement of computer technology and the huge growth of available data during the information age as well as the increased possibilites of storing large amounts of data has paved the way to use artificial neural networks to their full potential. The ability to quickly manage large amounts of data enables the training of neural networks, and in turn a whole new world of problem solving.

We have implemented neural networks to create an autonomous sign language interpreter for a subset of the American sign language (ASL) alphabet. The resulting application uses two neural networks; one for segmentation, which is able to determine the location of a hand in a picture, and one for classification, which is able to identify which sign the hand is showing.

The application delivers satisfactory results for most static signs for letters in the ASL alphabet. A graphical user interface was implemented as a user friendly product which allows the user to identify sign language using their web camera.

Sammandrag

Utveckling av kraftfullare datorer, tillgång till stora datamängder och ökad möjlighet att hantera stora datamängder har banat väg för att kunna utnyttja potentialen hos artificiella neurala nätverk. Möjligheten att snabbt kunna hantera stora mängder data för att träna ett neuralt nätverk att utföra en uppgift öppnar upp nya och spännande möjligheter för problemlösning.

Vi har utforskat möjligheten att använda neurala nätverk för att skapa en automatiserad teckenspråkstolk för ett urval av det amerikanska teckenspråket, ASL. Den applikation som tagits fram använder två neurala nätverk; ett segmenteringsnätverk som kan avgöra var i en bild händer finns och ett klassificeringsnätverk som kan identifiera vilket tecken en hand uppvisar.

Den applikation som utvecklats klarar med gott resultat av att identifiera de flesta statiska tecken för bokstäver i ASL. Ett grafiskt gränssnitt som hämtar bilder från en webbkamera och presenterar resultatet har tagits fram för att skapa en användarvänlig produkt.

Nyckelord: neurala nätverk, deep learning, maskininlärning, datorseende, teckenspråk, bildanalys, segmentering, klassificering, faltning

Förord

Arbetsprocessen genomfördes kollaborativt bortsett från ett fåtal delmoment. Den som fick tillgång till Purdue University:s teckenspråksdatabas och som extraherade bilder från videor i databasen var Anton Älgmyr. Det program som skrevs för maskskapande som använts för generering av träningsdata skrevs av Jacob Steffenburg. Erik Larsson och Jacob Steffenburg utvecklade tillsammans det användargränssnitt som används i slutprodukten. Skapandet av egna träningsbilder gjordes framför allt av Erik Larsson, Felix Viberg, Jacob Steffenburg och Svante Karlsson. Manuell maskning av de egenskapade bilderna gjordes framför allt av Felix Viberg och Svante Karlsson. Erik Larsson agerade även kontaktpunkt till den beräkningsdator som användes. Versionshantering och felsökning av git ansvarades för i huvudsak av Anton Älgmyr. En roll som TEXniker med kunskap om obskyr LATEX för rapportskrivande och formatering togs också av Anton Älgmyr. Vidare gjorde Anton Älgmyr merparten av alla rapportens figurer.

Arbetet var ett samarbete med en stor mängd öppna diskussioner där metoder och tillvägagångssätt debatterades. De som läste in sig i programmeringsbiblioteken, samlade data och läste teori delade viktiga insikter med varandra vilket bidrog till en kollektiv kunskapsutveckling. Erik Larsson, Felix Viberg och Svante Karlsson satte sig tidigt in i de relevanta biblioteken och skrev den kod som behövdes för att genomföra den första träningen av segmentering. Kod för träning av klassificeringsnätverk skrevs av Jacob Steffenburg och Svante Karlsson. Anton Älgmyr bidrog med algoritmer och programmering som förbättrade prestanda och körningstid för våra metoderna.

Redigering, formatering och sammanfogande av text till en sammanhängande helhet har utförts i grupp. Alla författare har haft betydande medverkan i samtliga kapitel.

Tack till

Vi vill tacka följande personer och organisationer för deras oumbärliga hjälp i arbetet:

Måns Larsson för stöd och feedback under arbetets och skrivandets gång. Tack även för trevliga fredagsmöten med gratis kaffe. Tack till *Institutionen för signaler och system* för utlåningen av superdatorn.

Calle Carlsson och *Andreas Eriksson* på fackspråk för värdefull respons på vårt skrivande.

Ronnie B. Wilbur och *Purdue University* för tillgång till deras teckenspråksdatabas.

Versionshanteringtjänsterna *Bitbucket* och *git* som höll arbetet utom kaos. *ShareLaTeX* som gjorde kollaborativ rapportskrivning till något hanterbart och inte mardrömsinducerande.

Annie Gjers, Oscar Johanneson och *Pauline Muñoz-Olsö* för teckenspråksdata.

Innehåll

1 Inledning **1**
1.1 Bakgrund . . . 1
1.1.1 Artificiella neurala nätverk . . . 1
1.1.2 Klassisk programmering kontra programmering genom maskininlärning . . . 2
1.1.3 Teckenspråk . . . 2
1.2 Uppgift och problem . . . 3
1.2.1 Målsättning . . . 3
1.2.2 Problem . . . 3
1.3 Avgränsningar . . . 4

2 Teori **5**
2.1 Artificiella neurala nätverk . . . 5
2.1.1 Framåtpropagation . . . 5
2.1.2 Felfunktion . . . 6
2.1.3 Bakåtpropagation . . . 6
2.1.4 Uppdatering av vikter . . . 7
2.1.5 Överträning . . . 7
2.1.6 Lagertyper . . . 7
2.1.6.1 Faltningslager . . . 7
2.1.6.2 Fullt kopplade lager . . . 8
2.1.6.3 Pooling-lager . . . 9
2.1.6.4 Aktiveringslager . . . 9
2.1.6.5 Soft max-lager . . . 9
2.1.7 Dropout-lager . . . 11
2.1.8 Batch normalization-lager . . . 11
2.2 Utvärdering av resultat . . . 11
2.2.1 Jaccard-index . . . 11
2.2.2 Förväxlingsmatris . . . 12
2.3 Bildbehandling . . . 12
2.3.1 Normalisering . . . 13
2.3.2 Färgbaser . . . 13
2.3.3 Skalinvarians . . . 13
2.4 Bildsegmentering . . . 14
2.4.1 Bildmasker . . . 14

3 Metod **15**
3.1 Arbetsprocess . . . 15
3.1.1 Införskaffande av data . . . 15
3.1.1.1 Purdue RVL-SLLL ASL . . . 15
3.1.1.2 Egengenererad data . . . 16
3.1.2 Skapande av segmenteringsmasker . . . 17
3.1.3 Förbehandling av bilder . . . 17
3.1.3.1 Byte av färgrymd . . . 17
3.1.3.2 Normalisering . . . 17
3.1.3.3 Jittering . . . 18
3.1.3.4 Padding av stående bilder . . . 18
3.1.3.5 Spegling . . . 18
3.1.4 Konstruktion av nätverk . . . 18
3.1.4.1 Konstruktion av segmenteringsnätverk . . . 19
3.1.4.2 Konstruktion av klassificeringsnätverk . . . 20
3.1.5 Träning . . . 20
3.1.5.1 Tränings- och valideringsdata . . . 21
3.1.5.2 Träning av segmenteringsnätverk . . . 22
3.1.5.3 Val av segmenteringsnätverk . . . 22
3.1.5.4 Träning av klassificeringsnätverk . . . 22
3.1.5.5 Val av klassificeringsnätverk . . . 23
3.2 Slutprodukt . . . 24
3.2.1 Användargränssnitt . . . 24
3.2.2 Segmentering . . . 24
3.2.2.1 Smart svepning . . . 25
3.2.2.2 Tröskelvärden . . . 26
3.2.2.3 Metoden findBiggestIsland . . . 26
3.2.3 Klassificering . . . 27

4 Resultat **28**
4.1 Slutprodukten . . . 28
4.1.1 Prestanda . . . 28
4.1.2 Evalueringstid . . . 28
4.2 Segmentering . . . 30
4.2.1 Tränings- och valideringsfel . . . 30
4.2.2 Utdata från olika lager . . . 30
4.2.3 Jaccard-index . . . 30
4.2.4 Tidsvinst från smart svepning . . . 33
4.3 Klassificering . . . 33
4.3.1 Tränings- och valideringsfel . . . 33
4.3.2 Sammanställning av förväxlingsmatriser . . . 33

5 Diskussion **35**
5.1 Segmentering . . . 35
5.1.1 Förbehandlingens betydelse . . . 35
5.1.2 Bildmaskernas kvalitet . . . 36
5.1.3 Tidsåtgång för segmentering . . . 36

5.2 Klassificering 36
5.2.1 Förväxling av tecken 36
5.2.2 Behovet av tre nätverk 37
5.3 Vidareutveckling av produkten 37
5.3.1 Snabbare evaluering 37
5.3.2 Mer träningsdata 38
5.3.3 Utökning av teckenvokabulär 38
5.3.3.1 Tecknande med båda händerna 39
5.3.3.2 Ickestatiska tecken 39
5.3.3.3 Tecknande framför ansiktet 39
5.3.4 Byte av programmeringsspråk 39

6 Slutsatser **40**

7 Bibliografi **41**

A ASL **I**

B Matematik **II**
B.1 Faltning II
B.1.1 Diskret faltning II
B.1.2 Flerdimensionell faltning II

C Förväxlingsmatriser **III**

D Nätverksstrukturer **VI**

E Evaluering av hela bilder med ett pixel-för-pixel-nätverk **VIII**
E.1 Problemet med nätverket VIII
E.2 En praktisk lösning IX

F Källkod **X**

1

Inledning

1.1 Bakgrund

Datorseende avser att lära datorer att känna igen och tolka digital bildinformation. Detta kan bland annat innebära personidentifiering ifrån ansiktsdrag och identifiering av hinder för en autonom robot. Ett relativt nytt [1] forskningsområde för realisering av datorseende är utveckling av artificiella neurala nätverk vilket till skillnad från klassisk programmering[1] tillåter datorer att genom träning[2] mot ett facit lära sig att känna igen mönster i information. Om nätverket tränats tillräckligt mycket för en viss uppgift kan det med samma beräkningar på ny indata dra slutsatser om denna, trots att det inte längre finns något facit att tillgå och att den nya indatan skiljer sig mot träningsdatan. Ett exempel på tillämpning är användning i självkörande bilar, med ett nätverk som lärt sig urskilja vägskyltar. Om nätverket exempelvis tränats till att identifiera stoppskyltar så kommer det förhoppningsvis detektera om det finns en stoppskylt i indatan, trots att den inte tränats med data där stoppskylten finns med på samma plats eller i samma vinkel och ljus. Andra exempel på tillämpningar kan vara att detektera ansikten i en bild eller detektera ett visst ansikte i en bild med flera ansikten.

1.1.1 Artificiella neurala nätverk

Artificiella neurala nätverk utvecklades då man försökte modellera hjärnans, eller mer specifikt neuronernas, sätt att bearbeta information. Warren McCulloch och Walter Pitts publicerade år 1943 en studie [2] som diskuterar vad författarna kallar *Threshold Logic* som bottnar i att informationsöverföring i biologiska eller artificiella neurala nätverk sker först då insignalen överstiger ett visst tröskelvärde. Denna studie lade grund för så väl forskning kring hjärnans biologiska processer som vidare utveckling av artificiella neurala nätverk. Andra tidiga insatser i forskningen kring artificiella neurala nätverk gjordes av Hebb under 1940-talet [1] (inlärningsprocesser), Farley och Clark 1954 [3] (simulation av Hebbska nätverk) samt Werbos 1975 [4] (bakåtpropagationsalgoritmen).

[1] Med klassisk programmering menas att skriva olika och specialiserade algoritmer för olika tillämpningar.

[2] Med träning menas i detta avseende att ett neuralt nätverk förses med träningsdata, exempelvis bilder på ett specifikt objekt, och ett facit som berättar i vilken del av bilden objektet återfinns. Med träning kan det neurala nätverket fås att identifiera objektet i främmande bilder.

Det fanns stor skepticism mot beräkningsmodellen under mitten och slutet av 1900-talet vilket berodde på att samtida datorkraft var alldeles för liten för att beräkningarna skulle kunna ge goda resultat. Intresset för artificiella neurala nätverk har dock ökat enormt allt eftersom teknikutvecklingen gått framåt och effektiv hantering av stora datamängder möjliggjorts. För att få goda resultat inom en rimlig tidsrymd är det av stor vikt för program baserade på neurala nätverk att träning sker på kraftfulla datorer med snabba processorer (och under senare år även grafikprocessorer), som kan hantera dessa stora datamängder.

1.1.2 Klassisk programmering kontra programmering genom maskininlärning

Idag kan maskininlärningsalgoritmer såsom de artificiella neurala nätverk bygger på överträffa både människan och klassisk programmering i viss form av problemlösning. Ett exempel på detta är det kinesiska strategispelet Go.

Go har länge varit ett av de abstrakta strategispel där datorer överträffats av människan och där utveckling av en framgångsrik självspelande dator ansetts prestigefyllt. Spelet tog över rollen från schack år 1997 efter att stormästaren Garri Kasparov besegrades av schackdatorn *Deep Blue* [5]. Skillnaden i komplexitet mellan Go och schack är enorm och antal möjliga drag vid ett givet tillfälle är ett tiotal i schack jämfört med ett hundratal i Go. Garri Kasparov kunde besegras tack vare en smart *brute force*-algoritm baserad på *alpha-beta pruning*, motsvarande algoritm kan inte användas framgångsrikt i Go.

Det tog till 2016 innan Googles AI *AlphaGo*, som baseras på neurala nätverk, lyckades besegra den japanska världsmästaren i Go - Lee Sedol [6]. En intressant aspekt i utvecklingen av *AlphaGo* är att träningen inte enbart skedde med förberedd data, utan också autonomt genom spelomgångar mot andra Go-datorer, människor och äldre versioner av sig själv.

Exemplet uppvisar en av de mest markanta skillnaderna mellan klassisk programmering och maskininlärning: att en maskininlärningsalgoritm kan träna på egen hand, utan att förlita sig på specifika instruktioner från en människa. Om inlärningen baserats på tillräckligt varierande data kan algoritmen också prestera väl mot mönster den inte stött på tidigare.

1.1.3 Teckenspråk

Teckenspråk är en grupp av språk där man till skillnad från tal- och skriftspråk kommunicerar via handgester där majoriteten av de som kan tala teckenspråk är döva eller stumma. Handgesterna kallas i dagligt tal för tecken och utförs statiskt eller ickestatiskt, med en eller båda händerna. Teckenspråk ser olika ut i olika delar av världen till följd av variationer i talspråk och grammatik och har, likt talspråk, dialektala skillnader.

1.2 Uppgift och problem

Vi har själva formulerat problem och målsättning för kandidatarbetet, fött ur nyfikenhet på neurala nätverk. Att arbetet inriktades på just teckenspråk beror på att det kan vara en intressant applikation ur bildanalytisk synpunkt. En fungerande teckenspråkstolk skulle dessutom vara ett användbart verktyg för kommunikation mellan teckenspråkstalande och icke teckenspråkstalande. Applikationer kan utvecklas för användning i utbildningssyfte eller som ett nytt sätt för teckenspråkskunniga att förmedla information.

1.2.1 Målsättning

Målsättningen var att implementera en teckenspråkstolk baserad på två neurala nätverk som från stillbilder urskiljer statiska tecken ur amerikanska teckenspråksalfabetet och översätter dessa till utskrift i text. Vidare beslutades att en applikation skulle utvecklas bestående av ett program körbart i Matlab där användaren tillåts ta bilder från sin webbkamera då användaren tecknar stationära tecken. Målsättningen var att programmet skulle kunna sköta alla steg från det att en bild tas till att motsvarande bokstav skrivs ut.

Det fastslogs utöver de konkreta målen även ett abstrakt mål med arbetet; utveckling av förståelse om hur artificiella neurala nätverk fungerar och vad de kan användas till.

1.2.2 Problem

För att ett nätverk skall kunna utföra en uppgift med goda resultat krävs det att nätverket byggs på rätt sätt och tränas med stora mängder data. Träningsresultaten kan förbättras om träningsdatan diversifieras till följd av att nätverket får fler fall att studera.

Ett utkast över hur programmet som projektet kretsar kring skapades med grund i sektion 1.2.1. Detta utkast kan i retrospekt specificeras av följande delproblem

- Insamlande av bildinformation från webbkamera
- Förprocessering av bildinformation med bland annat normalisering och byte av färgrymd
- Bildsegmentering av hand med hjälp av ett neuralt nätverk
- Förprocessering, kombinering av erhållen segmenteringsmask och ursprungsbild samt beskärning av resultatet
- Klassificering av tecken med hjälp av ett neuralt nätverk
- Utskrift i text

Hur delproblemen löstes presenteras i sektion 3.

1.3 Avgränsningar

Projektet begränsades till statiska tecken som ingår i alfabetet. Denna begränsning ledde till att programmet som utvecklades inte kan hantera bokstäverna *j* och *z*, som kräver dynamiska handrörelser, se Figur A.1 i Appendix A. Vidare ingår inga tecken som tecknas med två händer i alfabetet, eller tecken som tecknas framför ansiktet på teckenspråkstalaren.

2

Teori

Då detta arbete byggs kring neurala nätverk så kommer en stor del av teorin också kretsa kring detta. Vad är ett nätverk och hur är det strukturerat? Hur tränas och evalueras ett nätverk?

Utöver detta kommer det också presenteras teori för anknytande områden: metoder för att kunna utvärdera de system som utvecklas, bildbehandling för att förbereda data för nätverk samt en kort beskrivning av bildsegmentering och bildmasker.

2.1 Artificiella neurala nätverk

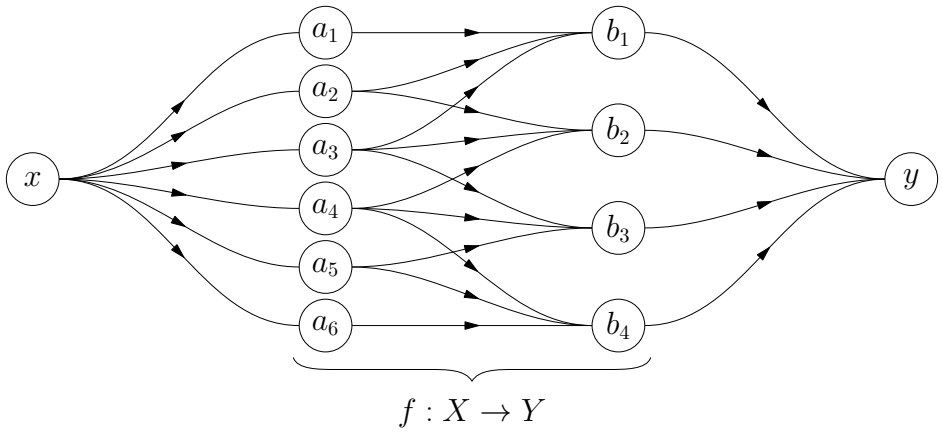
Artificiella neurala nätverk tar inspiration från biologiska neurala nätverk för att skapa ett system som kan tränas att utföra en uppgift. Den mest abstrakta beskrivningen av ett neuralt nätverk är ett system som definierar en funktion

$$f : X \to Y.$$

Funktionen f kommer i allmänhet vara en sammansättning av ett stort antal delfunktioner och exakt hur sammansättningen sker beror på hur nätverket är strukturerat. Ett typiskt sätt att representera ett neuralt nätverk är som en riktad graf, exempelvis som den graf som kan ses i Figur 2.1. Nätverket, som figuren illustrerar, kan ofta behandlas som en sammansättning av lager av neuroner där varje neurons funktion avgörs av en uppsättning vikter. Denna abstraktion av lager är mycket naturlig för många tillämpningar, exempelvis bildanalys. Val av upplägg och funktion (typ) hos dessa lager gör stor skillnad för hela nätverkets beteende och funktionsduglighet.

2.1.1 Framåtpropagation

När ett artificiellt neuralt nätverk evalueras skickas indata till det första lagret i nätverket. Datan behandlas lagervis och någon form av utdata produceras. Denna process kallas för *framåtpropagation*, indatan propagerar genom nätverket och transformeras med nätverkets delfunktioner, som illustreras i Figur 2.1.



Figur 2.1: En riktad graf som respresenterar ett neuralt nätverk med två lager. Varje nod i lagren har bidrag från kopplade noder åt vänster, och alla kopplingar har någon vikt (noden beskrivs av en funktion som beror på vikter samt kopplade nodvärden). Det är vikterna för kopplingar som avgör vilken sammansatt funktion, f, som systemet i stort utför.

2.1.2 Felfunktion

Under träningsfasen behövs ett mått på hur väl nätverket presterar, för att åstadkomma detta så definieras en felfunktion. En vanlig felfunktion är *mean squared error*

$$E(\mathbf{x}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - y_i^*)^2,$$

där n är antalet klasser, $\mathbf{x}$ är indatan, $\mathbf{y}$ är utdatan i form av sannoliketer och $\mathbf{y}^*$ är sanningen.

En annan vanlig felfunktion är *log-loss*, som definieras enligt

$$L(X, c) = -\log(P(X, c)),$$

där $P(X, c)$ är soft max-funktionen, noggrannare beskriven i sektion 2.1.6.5, X är uppskattade sannolikheter och c är sanningen.

2.1.3 Bakåtpropagation

Målet med att träna ett neuralt nätverk är att minimera dess felfunktion, därför är det av intresse att undersöka hur ändring av vikterna påverkar felet. *Bakåtpropagation* eller *Backward Propagation of Errors*, är en metod för att beräkna felfunktionens derivata med avseende på nätverkets vikter, $\frac{\partial E}{\partial w}$. Som namnet antyder stegar metoden baklänges och beräknar felfunktionens derivata lager för lager, där kedjeregeln möjliggör återanvändning av tidigare delresultat. Intuitivt kan denna propagationsriktning förklaras genom att ändring av vikter i ett senare lager (b i Figur 2.1) inte påverkar de tidigare lagrens (a i Figur 2.1) funktion. En mer ingående förklaring av bakåtpropagation kan läsas i manualen till Matconvnet [7].

2.1.4 Uppdatering av vikter

Det neurala nätverkets vikter uppdateras enligt en optimeringsalgoritm i ett försök att minimera felfunktionen. Inom maskininlärning används oftast optimeringsalgoritmen *stochastic gradient descent with momentum*, som opererar enligt

$$\Delta w_t = \mu \Delta w_{t-1} - \xi w_{t-1} - \frac{1}{B} \left(\frac{\partial E}{\partial w} \right)_{t-1} \tag{2.1}$$

$$w_t = w_{t-1} + \eta \Delta w_t \tag{2.2}$$

där η är *learning rate*, μ är *momentum rate*, ξ är *weight decay* och B är *batch size*. *Learning rate* avgör steglängden för uppdateringen, *momentum rate* avgör hur stor del av den tidigare uppdateringens riktning som ska inverka på den nya uppdateringen, *weight decay* avgör hur stor del av tidigare värden för vikter som ska glömmas och *batch size* är antalet *samples* (bilder, pixlar) som används vid en uppdatering.

2.1.5 Överträning

Ett problem som kan uppstå vid träning är att det neurala nätverket lär sig känna igen specifika exempel snarare än mönster i datan. Detta leder till att nätverken endast kan klassificera exempel som är snarlik träningsdatan.

2.1.6 Lagertyper

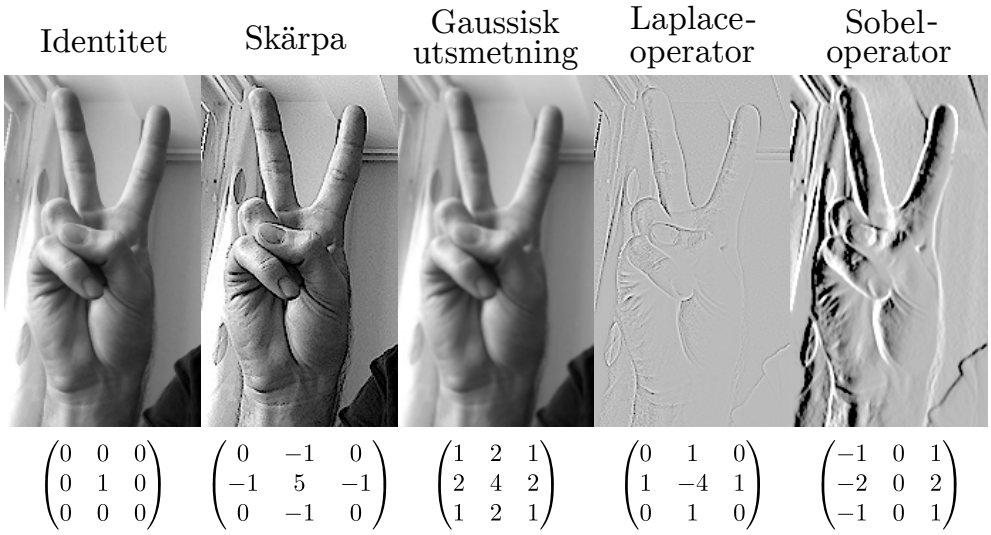
Neurala nätverk kan bestå av en mängd olika typer av lager. De typer som är relevanta för detta projekt kommer att presenteras i denna sektion. Gemensam notation för beskrivningen av lagren är: $P(x, y, d)$ som neuronen i föregående lager på plats (x, y) och djup d, $C(x, y)$ som neuronen i det lager som beskrivs, $W(x, y, d)$ som vikten i viktmatrisen på plats (x, y) och djup d. Ett subskript läggs till på viktnotationerna för att specificera om det gäller olika uppsättningar av vikter. I beskrivningar av lager antas att index x och y börjar från 0.

2.1.6.1 Faltningslager

Ett faltningslager (*convolutional layer*) är ett lager som utför en diskret faltningsoperation. En kort beskrivning av faltning, från det kontinuerliga endimensionella fallet till det diskreta flerdimensionella fallet, kan hittas i appendix B. I specialfallet bildanalys kommer ofta en tvådimensionell (eller tredimensionell faltning med samma djup som föregående lager) att göras. I fallet med en faltning med en $s \times s \times d$-matris så ser relationen mellan lager ut enligt

$$C(i, j) = \sum_{x,y \in M} \sum_{d} W(x, y, d)\, P(i + x, j + y, d), \quad M = \left\{ x, y : -\left\lfloor \frac{s}{2} \right\rfloor \leq x, y \leq \left\lceil \frac{s}{2} \right\rceil \right\},$$

där den något omständiga definitionen av M är sådan att (i, j) alltid ligger i mitten av (eller något upp till vänster av) området $(i, j) + M$. Neuroner *delar i detta fall vikter* för faltningarna, så antalet vikter i lagret är bara beroende av



Figur 2.2: En bild med fem olika 3×3 faltningsmatriser applicerade. Figuren ämnar påvisa bredden och kraften i att beskriva operationer med hjälp av faltningsmatriser, och kopplingen mellan faltning och bildanalys. En kort beskrivning av operationerna följer. Identiteten är den diskreta motsvarigheten till ett Diracdelta som lämnar bilden oförändrad. Skärpa kan fås genom den andra masken, som framhäver pixeln i mitten där masken appliceras samtidigt som likheter med grannar förtrycks. Den tredje bilden är ett typexemepel på en bildhanteringsoperation som på engelska kallas *Gaussian blur*, namnet kommer från att faltningsmasken approximerar en Gaussfördelning. Den fjärde bilden är en variant av en diskret Laplaceoperator som är användbar då den framhäver kanter. Slutligen är Sobeloberatorn en variant av en derivata på bilden, värt att notera är att operatorn tenderar att på ljussatta bilder framhäva kurvatur hos ett objekt.

storleken på faltningsmatrisen. Ett schematiskt exempel för ett faltande system med faltningsmatris av storlek $(5 \times 5 \times 1)$ kan ses i Figur 2.3.

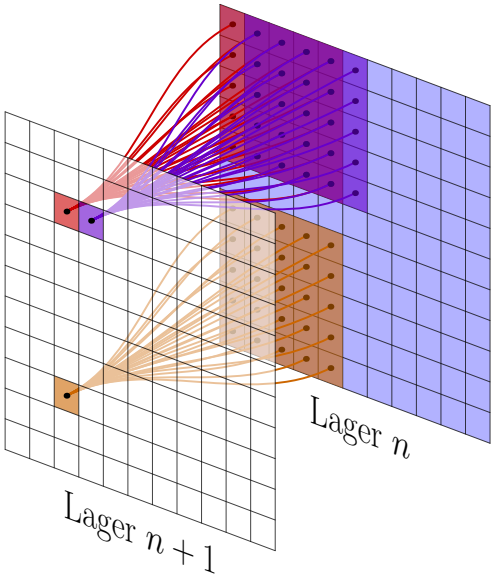
Vad ett faltningslager kan göra, och dess koppling till bildanalys och bildhantering, visas här med ett exempel. I Figur 2.2 ses ett fåtal möjliga operationer där en faltningsmatris med storlek $(3 \times 3 \times 1)$ används. Faltning är, som figuren visar, ett kraftfullt verktyg för att operera på bilddata.

2.1.6.2 Fullt kopplade lager

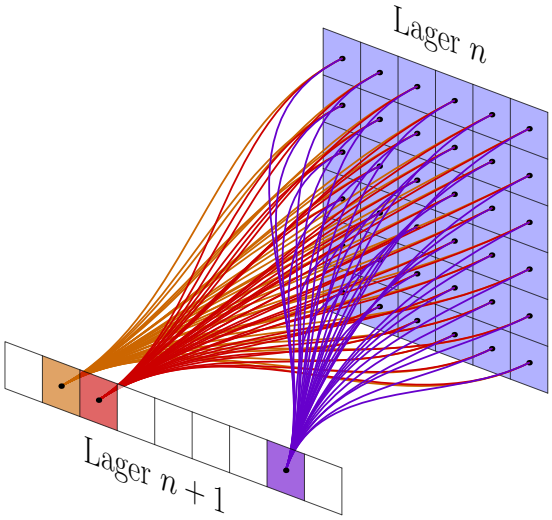
Ett fullt kopplat lager (*fully connected layer*) är ett lager där alla neuroner i ett lager är kopplade till alla neuroner i det föregående lagret, se Figur 2.4. Detta är ett specialfall av ett faltningslager där faltningsmatrisen är lika stor som indatan och där utdatan från faltningen då blir en enda neuron,

$$C(i,j) = \sum_x \sum_y \sum_d P(x,y,d)\, W_{i,j}(x,y,d).$$

På grund av att varje neuron kan plocka värden från samtliga neuroner i det tidigare lagret så kan globala särdrag urskiljas, vilket passar väl för klassificering. Värt att notera är att då varje neuron kopplas globalt så växer antalet kopplingar (vikter) snabbt med antal neuroner i det fullt kopplade lagret, så lager med alltför många fullt kopplade neuroner är ovanliga.



Figur 2.3: Ett faltningslager med en 5×5 faltningsmatris med tre neuroners kopplingar inritade.



Figur 2.4: Ett fullt kopplat lager med tre neuroners kopplingar inritade.

2.1.6.3 Pooling-lager

Ett pooling-lager (pooling layer) används för att reducera datamängder och för att sålla bort data. Detta görs genom att dela upp datan i områden och låta något mått representera varje område. Det finns flera olika varianter, men en av de vanligaste är max pooling, där det är maxvärdet i området som tillåts propagera. Det är också denna typ av lager som används i projektet. Andra typer av pooling-lager är min pooling och mean pooling. Ett tvådimensionellt max pooling-lager som reducerar lagerstorlek med en faktor $1/n$ i varje riktning (reducerar antalet neuroner med en faktor $1/n^2$) kan beskrivas enligt

$$C(i,j,d) = \max_{\substack{n\,i \leq x < n\,(i+1) \\ n\,j \leq y < n\,(j+1)}} P(x,y,d).$$

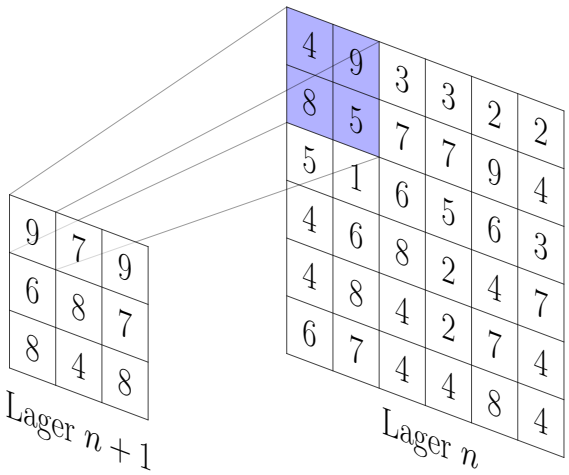
Ett exempel på ett max pooling-lager (med lager av djup 1) kan ses i Figur 2.5.

2.1.6.4 Aktiveringslager

Ett nätverk utan aktiveringslager kan (utöver pooling) enbart utföra linjära operationer. För att införa icke-linjära egenskaper så tillämpas ofta aktiveringslager som utför en funktion f på alla neuroner. Val av aktiveringfunktion beror på tillämpning, men ett typiskt val är en så kallad *rectified linear unit* (ReLU). Ett urval av aktviveringsfunktioner, med definitioner och grafer presenteras i Tabell 2.1.

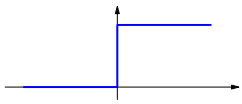
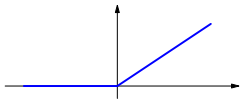
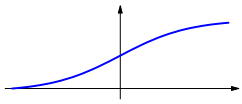
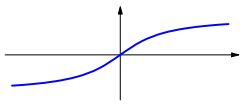
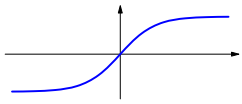
2.1.6.5 Soft max-lager

Det är ofta fördelaktigt i ett klassificeringsnätverk att ha normerad utdata sådan att den kan tolkas som en sannolikhetsfördelning. Det finns många sätt att normera



Figur 2.5: Ett max pooling-lager där storleken halveras i både x och y led. Den markerade datapunkten längst upp till vänster i lager $n+1$ blir maxvärdet för det markerade området i lager n.

Tabell 2.1: Ett urval aktiveringsfunktioner med tillhörande definitioner och grafer.

Namn	**Funktion**	**Graf**
Styckvis konstanta/linjära		
Stegfunktion	$f(x) = \begin{cases} 0 & x < 0 \\ 1 & x \geq 0 \end{cases}$	
Rectified linear unit	$f(x) = \begin{cases} 0 & x < 0 \\ x & x \geq 0 \end{cases}$	
Sigmoider (S-lika kurvor)		
Logistisk funktion	$f(x) = \dfrac{1}{1 + \mathrm{e}^{-x}}$	
Invers tangens	$f(x) = \tan^{-1} x$	
Tangens hyperbolicus	$f(x) = \tanh x$	

data, en typ som ofta används inom inlärning utgår från en så kallad softmax-funktion[8],

$$\text{softmax}(\mathbf{x}, i) = \frac{\mathrm{e}^{x_i/\tau}}{\sum_{a \in x} \mathrm{e}^{a/\tau}},$$

där τ kallas för en ”temperaturparameter”. Läsare som känner till statistisk fysik kopplar nog snabbt detta till Boltzmannfördelningen, där $\tau = k_B T$ och x är energier i ett system.

Fördelen med denna typ av normering framför den typiska normeringen $x_i/(\sum_{a \in x} a)$ är att stora värden dominerar och små värden förtrycks så att fördelningen ofta (i de fall där det finns ett klart största värde) approximerar en indikatorfunktion (ett diskret Diracdelta).

2.1.7 Dropout-lager

Dropout-lager fungerar som ett filter som blockerar vissa neuroner under träning av ett nätverk. Det låter negativt att begränsa informationsflödet i ett nätverk men effekten blir att noder som tidigare förlitat sig på utdata från en nod behöver anpassa sig när denna nod blockeras. Användandet av dropout-lager tvingar alltså nätverket att lära sig att analysera data på olika sätt och att inte förlita sig på att hitta specifika inslag, vilken i sin tur motverkar överträning.

2.1.8 Batch normalization-lager

Batch normalization används under träning av neurala nätverk och utför normalisering över varje kanal för utdatan från lagret innan. Detta har visat sig ge fördelar såsom att möjliggöra högre *learning rates*, vilket medför färre träningssteg för att nå resultat. [9]

2.2 Utvärdering av resultat

Beroende på vad ett system har för uppgift finns olika typer av mått som kan nyttjas för att mäta dess prestanda. Ett segmenteringssystem kan med fördel utvärderas med måttet *Jaccard-index*, som presenteras i sektion 2.2.1. Ett klassificeringssystem kan utvärderas med en *förväxlingsmatris*, vars teori redogörs för i sektion 2.2.2.

2.2.1 Jaccard-index

Jaccard-index, α, är ett sätt att mäta hur väl två mängder A och B överlappar. Indexet beräknas enligt

$$\alpha = \frac{|A \cap B|}{|A \cup B|},$$

där nämnvärda specialfall (förutsatt att mängderna är icke-tomma) är då $A \cap B = \varnothing$ (disjunkta mängder) där $\alpha = 0$ och då $A = B$ (identiska mängder) där $\alpha = 1$.

Jaccard-index kan exempelvis användas för att mäta precisionen för ett segmenteringssystem, där

$$A = \{\text{Index } (i, j) \text{ för målobjektet i den sanna segmenteringsmasken}\}$$
$$B = \{\text{Index } (i, j) \text{ för målobjektet i systemets gissning}\}.$$

2.2.2 Förväxlingsmatris

För att utvärdera ett klassificieringssystems prestanda kan en så kallad förväxlingsmatris (*confusion matrix*) studeras. En förväxlingsmatris F har storleken $n \times n$, där n är antalet klasser, och innehåller information om antalet korrekta och felaktiga klassificeringar som utförts på ett dataset.
Element f_{ij} anger antalet objekt av klass i som klassificerats som ett objekt av klass j. Ett högpresterande klassificeringssystems förväxlingsmatris har således få nollskilda element utanför diagonalen.

Den totala precisionen för klassificieringssystemet kan beräknas enligt

$$p^* = \frac{\sum_i f_{i,i}}{\sum_{i,j} f_{i,j}}.$$

En förväxlingsmatris ger även mer detaljerad information om hur väl enskilda klasser behandlas av systemet. Känsligheten k_i anger hur stor andel av objekt tillhörande klass i som har klassificerats korrekt och beräknas enligt

$$k_i = \frac{f_{i,i}}{\sum_j f_{i,j}}.$$

Felklassificeringsandelen FKA_i ger andelen objekt som felaktigt klassificerats som i och beräknas enligt

$$\text{FKA}_i = \frac{\sum_{j \neq i} f_{j,i}}{\sum_j f_{j,i}}.$$

2.3 Bildbehandling

Strävan efter att ett nätverk skall kunna hantera så många olika typer av bilder som möjligt kräver att nätverket ska kunna hantera suboptimala fall. Exempel på detta kan vara stor ljusintensitetsvarians mellan olika bilder eller att det sökta objektet för ett segmenteringsnätverk har olika storlek. Det naiva sättet att hantera dessa problem på är att låta nätverket själv lära sig att ignorera dessa faktorer under träningen. Lämnas indatan oförändrad ökar dock risken för att problem uppstår. Orsaker till detta kan bland annat vara att ljusintensitet inte återskapas vidare bra i RGB-värden, som innehåller värden för mängden rött, grönt och blått för en viss pixel.

Eftersom det kan vara både kostsamt och tidsödande att samla in ny data för träning kan det vara bättre att försöka diversifiera den data som redan finns. Bildbehandling syftar till att åstadkomma just detta; att genom smarta manipulationer av bilddata

antingen differentiera träningsdatan så att den bättre reflekterar verkligheten, eller att göra data som skall utvärderas mer lik den träningsdata som nätverket har lärt sig hantera.

2.3.1 Normalisering

Normalisering sker för att minska variationer i indatan, om detta inte utförs finns en risk att nätverket överkompenserar (eller underkompenserar) inlärningen av vissa inslag. Normaliseringen sker enligt

$$y = \frac{x - \mu}{\sigma} \tag{2.3}$$

där y är utdata, x är indata, μ är medelvärde och σ standardavvikelse. Detta ger en fördelning av värden som har medelvärde 0 och standardavvikelse 1.

2.3.2 Färgbaser

Inom maskininlärning spelar valet av färgbas stor roll, likt hur valet av bas spelar stor roll för linjär algebra. Den typiska färgbasen inom digitala bilder, RGB (röd, grön, blå), passar ofta dåligt för bildanalys då kanalerna inte representerar någon egenskap av större intresse. Det finns på grund av detta anledning att övergå till en färgbas som bättre representerar intressanta egenskaperna hos systemet. Exempel på sådana färgbaser är XYZ, LUV och LCH.

XYZ (CIE 1931)[10] är en systemoberoende[1] färgbas konstruerad efter empirisk data för att efterlikna hur människan uppfattar färger. Denna bas ligger sedan till grund för de sistnämnda två färgbaserna. LUV transformerar om XYZ så att ljusintensitet separeras till sin egen kanal L (luminance). LCH går ytterligare ett steg och gör om LUV till cylindriska koordinater där höjd är L, radien är färgstyrkan C (chroma) och fasen är kulören H (hue).

Att använda LUV/LCH framför XYZ leder till att ljusstyrkan kan behandlas separat, vilket ofta är fördelaktigt vid bildanalys.

2.3.3 Skalinvarians

Skalinvarians kan vara viktigt om man inte vill att nätverket skall lära sig att objekt har en viss storlek för all indata. Om träningsdatan är homogen i detta avseende, det vill säga att storleken på det sökta objektet har samma dimension i alla bilder, och man förväntar sig att detta inte är sant för data på vilka nätverket skall appliceras behövs träningsdatan justeras. Enklast görs detta genom en att slumpmässigt beskära bilderna kring målobjektet. När datan inte längre uppvisar något mönster i storlek på målobjektet så kommer nätverket ej heller att lära sig att leta efter detta. Att slumpmässigt beskära träningsdata är en del av en metod som kallas jittering. I jitteringen ingår, till exempel, att rotera, skjuva och translatera träningsdata för

[1] En färg ser identisk ut oavsett vilket system (monitor, skanner, kamera) som den visas på, detta åstadkoms genom att korrigera färgrymdens vitpunkt efter systemet.

att förbereda nätverken på flera fall och därmed erhålla egenskaper som rotations-, skjuvnings och translationsinvarians.

2.4 Bildsegmentering

För att ett neuralt nätverk ska lära sig klassificera indata krävs det att den vid träning får veta vad rätt svar är, då den i efterhand behöver något att förhålla sig till när vikterna skall justeras via *stochastic gradiant descent*, beskrivet i sektion 2.1.4. Detta facit ges enkelt till ett nätverk som ska klassificera objekt, då det endast handlar om att sätta någon form av etikett på träningsdatan. Om nätverket istället används för att segmentera en hel bild så behövs en etikett för varje pixel, då varje enskild pixel måste klassificeras som antingen en del av målobjektet eller ej. Detta löses genom bildmasker, eller *segmenteringsmasker*.

2.4.1 Bildmasker

Bildmasker (eller kortare masker) används för att representera existensen av ett objekt i en bild. Typiskt är att använda en binär mask där 1 representerar existensen av ett objekt i en specifik pixel och 0 representerar avsaknaden av detta.

Det går även att ha en mask med kontinuerlig skala (ex. för att representera sannolikhet att något är del av ett objekt) eller en flertrösklad skala där olika värden motsvarar olika typer av objekt.

3

Metod

Det arbete denna rapport beskriver är skapandet och träningen av två nätverk som samverkar för att givet en bild skriva ut vilket teckenspråkstecken som förekommer i bilden. Ett första nätverk tränades till att producera segmenteringsmasker och ger som utdata information om var i en bild en eventuell hand befinner sig. Ett andra nätverk tränades för att givet en bild med tillhörande mask kunna avgöra vilket tecken, om något, handen tecknar.

Ett verktyg som använder dessa nätverk sammanställdes i form av ett användargränssnitt där användaren tillåts evaluera en bild tagen med webbkamera. I detta användargränssnitt utförs hela evalueringsprocessen genom implementation av de båda nätverken samt användning av andra metoder som syftar till att både sänka tidsåtgången och förbättra kvalitén på utdata.

3.1 Arbetsprocess

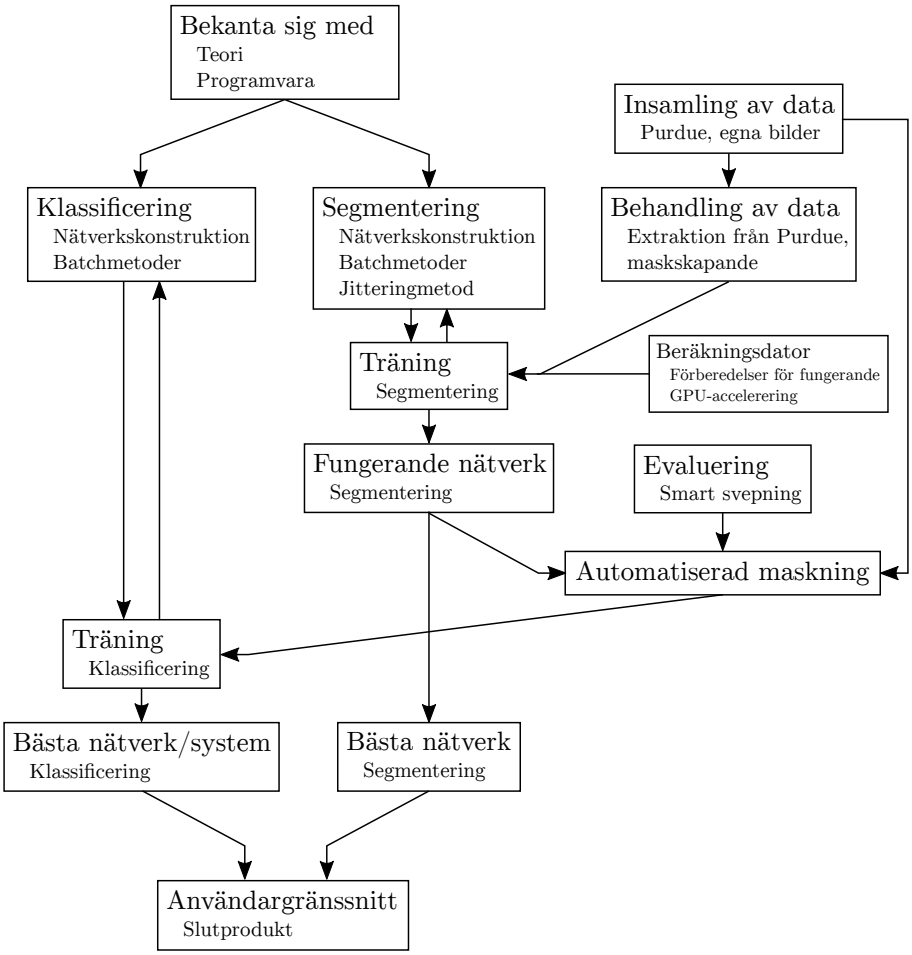
Följande sektioner beskriver den arbetsgång som lett till en färdig slutprodukt. En schematisk översikt över arbetsflödet presenteras som flödesschemat i Figur 3.1.

3.1.1 Införskaffande av data

För att få en stor mängd träningsdata där olika typer av ljussättningar och miljöer täcks in, samlades träningsdata in från olika källor. Den största delen av datan genererades manuellt, där förloppet presenteras i sektion 3.1.1.2. Data extraherades även från *Purdue RVL-SLL ASL American Sign Language Database* [11], samt hämtades från den färdiga databasen *Hand Gesture Recognition* [12, 13, 14, 15].

3.1.1.1 Purdue RVL-SLLL ASL

En del av den data som användes kommer från Purdue RVL-SLLL American Sign Language Database [11]. Databasen innehåller ett stort antal videoklipp där olika teckenspråksanvändare visar ett flertal olika tecken. För att kunna använda denna data behövde videor beskäras till mindre klipp som innehåller enbart en bokstav vardera. Videor klipptes så att alla bildrutor i de klippta videorna innehöll tidpunkter då tecknet utfördes, övergångarna mellan tecken sållades alltså bort. Klippning av video gjordes enklare genom att skriva skript för den mediaspelare som användes,



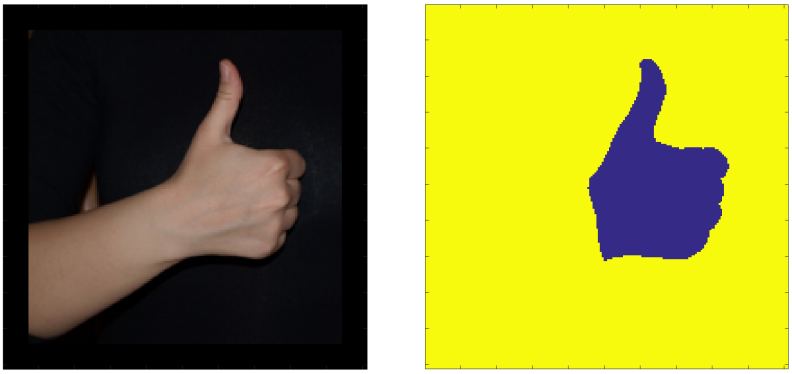
Figur 3.1: Flödesschema som beskriver arbetsgången från början av arbetet till slutprodukten.

specifikt *luaskript* till *mpv*. Skriptet skapade kortkommandon som markerade start och slut på sekvensen och spelade upp det klipp som valts. Ytterligare ett kortkommando användes för att acceptera klippet och skriva timestamps till en index-fil.

När väl index för alla videor skapats så skrevs skript för att i batch extrahera filmklipp utgående från indexet (*bashskript* som använde sig av *ffmpeg*). Ytterligare ett skript gick sedan över de extraherade filmklippen och extraherade bildrutor som skrevs till bildfiler.

3.1.1.2 Egengenererad data

En databas med ungefär 2500 egentagna bilder skapades i ett försök att täcka in olika fall av ljussättning och bakgrunder som kan tänkas förekomma vid användning av slutprodukten. Dessa bilder har använts som tränings- och valideringsdata men också för att genomföra tester av nätverken. Bilderna togs med webbkamera av samma typ som slutprodukten utvecklades för.



Figur 3.2: Figuren visar en bild med tillhörande mask. Tillsammans bildar detta par träningsdata till segmenteringsnätverket.

3.1.2 Skapande av segmenteringsmasker

För att träna segmenteringsnätverket krävdes masker för varje bild. Maskerna konstruerades som binära matriser med samma dimension som bilderna och består av områden med ettor och nollor där området med ettor beskriver var i bilden handen befinner sig. Exempel kan ses i Figur 3.2.

För att producera masker till träningsdatan konstruerades *MaskMaker*, ett enkelt bildbehandlingsprogram skrivet i Matlab, som låter användaren markera kanten på ett objekt, i detta fall en hand. *MaskMaker* genererar därefter en till ursprungsbilden tillhörande binär segmenteringsmask med samma dimension som bilden.

3.1.3 Förbehandling av bilder

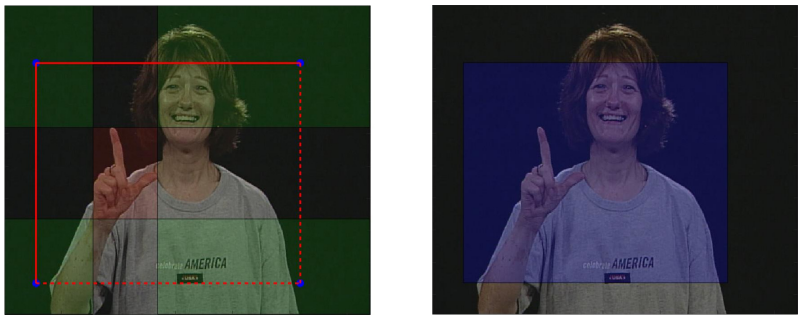
För att göra det enklare för nätverken att dra liknande slutsatser för bilder tagna under skilda förhållanden krävs förprocessering som gör evalueringsdatan mer homogen. Exempelvis bör bilder på ett objekt tagna i solljus generera samma resultat som bilder på samma objekt tagna i ett rum med låg ljusintensitet. Den förprocessering som utvecklades under arbetet med teckenspråkstolken beskrivs i denna sektion.

3.1.3.1 Byte av färgrymd

För färgrymdstransformationen används Matlabpaketet *Colorspace Transformations*[16] av Pascal Getreuer. Transformationen görs från RGB, med en färgkanal per lager, till LUV, med ljusintensitet i det första lagret och färgdata i det andra och tredje lagret. Anledningen till detta var att nätverket separat skulle kunna behandla färg och intensitet, vilket visade sig vara fördelaktigt för nätverket då det enklare kunde utröna vad hudfärg borde vara.

3.1.3.2 Normalisering

Då utseendet av den använda träningsdatan varierade kraftigt beslutades att normalisering skulle ske i alla lager för samtliga bilder. Om träningsdatan hade varit mer homogen skulle annars samma normalisering kunna göras på alla bilder samtidigt.



Figur 3.3: Figurerna visar implementation av jittering för att erhålla skalinvarians. Startpunkt väljs ut i det övre vänstra området och därefter slumpas en tillåten rektangel med samma proportioner som bilden fram. Bilden beskärs till rektangeln och kvar blir området markerat med blått.

Skillnaden blir att μ och σ i (2.3) beräknas per lager i varje bild, snarare än per lager för alla bilder sammantaget. Normalisering av bilder görs vid evaluering av bilder och gjordes vid träning av båda nätverken.

3.1.3.3 Jittering

För att utöka mängden träningsdata skrevs en funktion som kapar bort en slumpmässig mängd bildinformation i två dimensioner med bibehållna proportioner utan att skära bort objektet, i detta fall handen. Funktionen skrevs för användning på träningsdatan för att simulera att olika användare håller handen vid olika avstånd från kameran. Hur jitteringmetoden fungerar visualiseras i Figur 3.3.

3.1.3.4 Padding av stående bilder

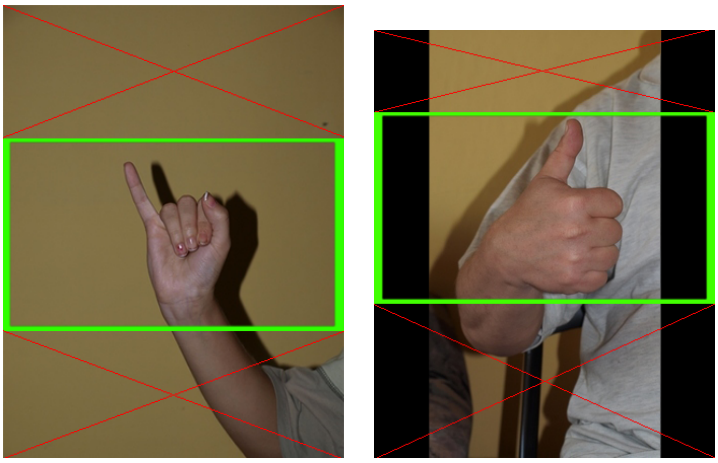
Då träningsdatan ursprungligen bestod av bilder i både stående och liggande format skrevs en funktion som kapar bort en bestämd mängd bildinformation (se Figur 3.4a), eller fyller ut med en bestämd mängd bildinformation (se Figur 3.4b), skild från objektet, så att alla bilder får ungefär samma proportioner. Detta gjordes för att motverka att kontraktionsavbildning sker i olika dimensioner för olika bilder vid nedskalning till kvadratisk form. Effekten av felaktig kontraktionsavbildning kan ses i Figur 3.5.

3.1.3.5 Spegling

Alla bokstäver i ASL tecknas spegelvänt av vänster och högerhänta. I databaserna förekom bilder på vänsterhänta teckenspråksanvändare, dessa bilder fick speglas för att matcha övrig träningsdata. Spegligen medför ingen större begränsning då inkommande evalueringsbilder kan speglas på samma sätt.

3.1.4 Konstruktion av nätverk

MatConvNet är det paket för Matlab, med funktioner som låter användaren konstruera faltningsbaserade neurala nätverk, som använts under projektet [7]. MatConvNet är integrerat i VLFeats öppna biblioteket för datorseende och flera av de funktioner som ingår i detta bibliotek har också använts under projektet [17].



(a) Beskuren **(b)** Paddad och beskuren

Figur 3.4: Figuren visar hur bilder blir beskurna eller paddade och beskurna för att omvandla stående format till liggande i syfte att motverka felaktig kontraktionsavbildning.

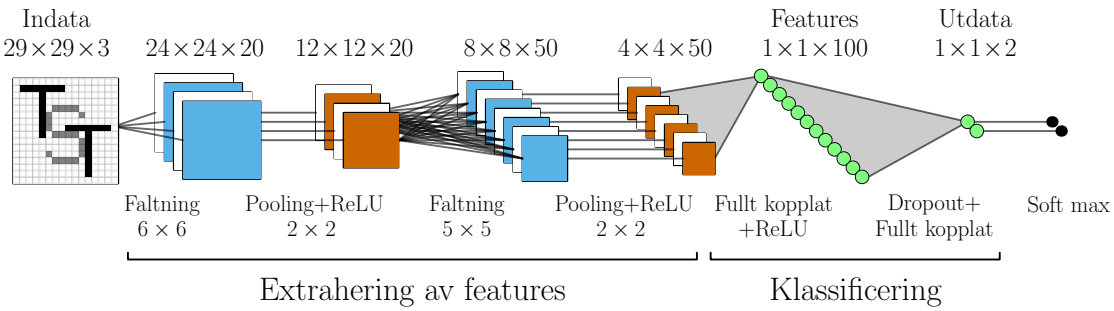
Figur 3.5: Figuren visar resultatet av att skala ner stående och liggande bilder till kvadratisk form. Att använda bägge dessa typer av felaktig skalning som träningsdata vore att träna nätverket mer än nödvändigt för att identifiera samma tecken. Stående bilder i träningsdatan har därför blivit beskurna enligt Figur 3.4.

Utgångspunkten för strukturen av både segmenteringsnätverket och klassificeringsnätverket var nätverket *cnn_mnist*, som är ett exempelnätverk som följer med MatConvNet-paketet. Detta är ett klassificeringsnätverk som är konstruerat för att klassificera bokstäver ur bilder. Grundstrukturen är att faltningslager varvas med *max pooling*-lager, och med ett aktiveringslager mellan *fully connected*-lagren. Som felfunktion används *log-loss*, se sektion 2.1.2.

3.1.4.1 Konstruktion av segmenteringsnätverk

De nätverk som konstruerades utför pixelvis segmentering där indatan är ett område ($29 \times 29 \times 3$) kring en pixel och utdatan är den beräknade sannolikheten för att pixeln är del av en hand. Av sex segmenteringsnätverk som konstruerades har fyra den struktur som kan ses i Figur 3.6. De två övriga nätverken skiljer sig enbart i att de saknar dropout-lager.

För att få dimensionerna på viktmatriserna att stämma överens med den aktuella



Figur 3.6: Diagram som visar struktur hos det använda segmenteringsnätverket. Diagrammet är avsmalat då uppgreningen för varje lager kan vara stort. De faktiska storlekarna (bredd × höjd × antal lager/maps) kan ses över varje steg.

indatan användes kommandot *vl_simplenn_display* som utför beräkningarna enligt den angivna strukturen och visar storleken på utdatan. Ett antal försök gjordes för att förbättra den ursprungliga strukturen beskriven under sektion 3.1.4, bland annat provades att minska det totala antalet vikter för att snabba upp nätverket men detta gav större negativa effekter än positiva och det mindre nätverket kunde inte fås att segmentera bilder korrekt. Försök gjordes med användning av *batch normalization* men utan goda resultat. *Dropout*-lager lades till mellan *fully connected*-lagren vilket visade sig fungera väl i vissa fall, och då främst för komplicerade tecken med hög detaljrikedom. Slutprodukten använder *dropout*-lager men inte *batch normalization*.

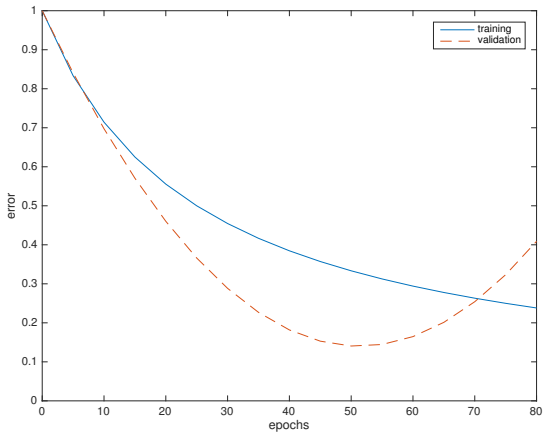
3.1.4.2 Konstruktion av klassificeringsnätverk

För konstruktionen av klassificeringsnätverket antogs att segmenteringsnätverk och klassificeringsnätverk har många likheter varför samma modell användes. De enda betydande skillnaderna mot segmenteringsnätverket var att viktmatrisernas storlekar anpassades för indata av dimension $64 \times 64 \times 4$, samt att antalet klasser utökades. Istället för två klasser (hand, icke hand) som i segmenteringsnätverket, konstruerades nätverket för en klass per tränad bokstav. Lagrena i indatan representeras av tre kanaler för bilden i LUV-värden och en kanal för den genererade masken från segmenteringsnätverket. Klassificeringsnätverket liksom segmenteringsnätverket utnyttjar både *dropout*-lager och lager för *batch normalization*. Den specifika strukturen finns beskriven i Tabell D.1 i Appendix D.

3.1.5 Träning

Merparten av träningen utfördes på en fjärrstyrd dator[1] som utlånades av institutionen för Signaler och System. Detta visade sig bli en mycket stor tillgång för arbetet då datorn hade tillräckligt hög prestanda för att ge träningsresultat relativt snabbt vilket innebar att tester av parameterinställningar kunde göras i högre frekvens. För träning användes funktionen *cnn_train* från MatConvNet som tar ett initialiserat

[1] Intel i7-5930K, GTX Titan X 12GB, 32GB RAM



Figur 3.7: Exempel på hur valideringsdata kan visa att ett nätverk överträtnats. Vid epok 50 börjar valderingsfelet öka samtidigt som träningsfelet sjunker. Nätverket blir alltså bättre på att tolka träningsdata men sämre på att tolka främmande data.

nät, ett dataset och en *getBatch()*-metod[2] som indata. För att nå ett något sånär godtagbart resultat behövde segmenteringsnätverket slutligen tränas i omkring 20 epoker med runt 2000 bilder, där en träningsepok tog ungefär sex timmar att genomföra. Motsvarande träning på en vanlig dator skulle ta en storleksordning längre tid. Förutom detta gav användning av den kraftfullare datorn möjlighet att träna flera nätverk samtidigt utan att tidsåtgången ökade nämnvärt.

Ett nätverks träning är uppdelat i epoker. Att en epok är slutförd innebär att all träningsdata har passerat nätverket. Epoker sker stegvis då nätverken ges en delmängd av träningsdatan i vad som kallas för en batch. Dessa väljs ut slumpmässigt men inga upprepningar av träningsdata förekommer under en epok.

Efter att en batch har behandlats sker bakåtpropagation och viktmatriserna uppdateras via *stochastic gradient descent*, som beskrivs i sektion 2.1.4.

3.1.5.1 Tränings- och valideringsdata

För att få ett mått på hur väl nätverken tränats avsattes en femtedel av all data till validering. Valideringsdatan användes alltså inte för träning utan enbart för utvärdering av nätverket mellan varje träningsepok.

Det visade sig användbart att få ett mått på hur väl ett nätverks prestanda utvecklades för främmande data då det blev enkelt att se om parametrar behövde justeras eller om nätverket blev överträtnat. *Valideringsfelet* är dessutom ett enkelt sätt att jämföra olika nätverk. Ett exempel på hur valideringsfelet kan användas för att se när ett nätverk är överträtnat visas i Figur 3.7.

[2] *getBatch* plockar ut en delmängd ur datan samt rätt etiketter för dessa. Vid träning går det att bestämma storleken på batchen, vilket styr mängden data som ska användas vid en uppdatering av viktmatriserna.

Tabell 3.1: De sex utvalda nätverk vars Jaccard-index jämfördes (se sektion 2.2.1). Resultatet presenteras i Tabell 4.2 samt i Figur 4.5b. Dataseten är ASL (*Purdue University*), HGR (*Hand Gesture Gecognition*) och TST (egengenererad data).

Nätverksnamn	Dataset	Learning rate	Dropout	Epoker
PurdueNet	ASL + HGR	0.001		22
18Amigos	TST	0.001	×	33
BonJoviLives	ASL + HGR	0.001	×	15
Monznet	ASL + HGR + TST	0.001		50
Adderall	ASL + HGR + TST	0.001	×	50
CleganeBowl	ASL + HGR + TST	0.0001	×	77

3.1.5.2 Träning av segmenteringsnätverk

Inför träning av segmenteringsnätverket lagrades alla träningsbilder och målvärden i flerdimensionella matriser. Träningen i sig gick ut på att låta nätverket löpa igenom alla pixlar på alla bilder i slumpmässig ordning och uppdatera viktmatriserna för att få resultatet att närma sig bifogade målvärden. För att nätverket skulle kunna undersöka pixlar på kanten av en bild lades en 14 pixlar bred ram till bilderna [3].

Att producera ett väl fungerade segmenteringsnätverk visade sig vara tidsödande. Till en början användes endast de bilder som var försedda av Purdue University, se sektion 3.1.1.1. Nätverket gav godtagbara resultat för de bilder som åsidosatts för validering av nätverket, som alltså kommer från dataset med bilder tagna under samma förutsättningar, men det fungerade sämre för bilder vi själva producerat.

De egengenererade bilderna, se sektion 3.1.1.2, kombinerade med bilderna från Purdue University och bilderna från Hand Gesture Recognition-databasen resulterade slutligen i ett nätverk bra nog för att kunna ta fram träningsdata för klassificeringsnätverket.

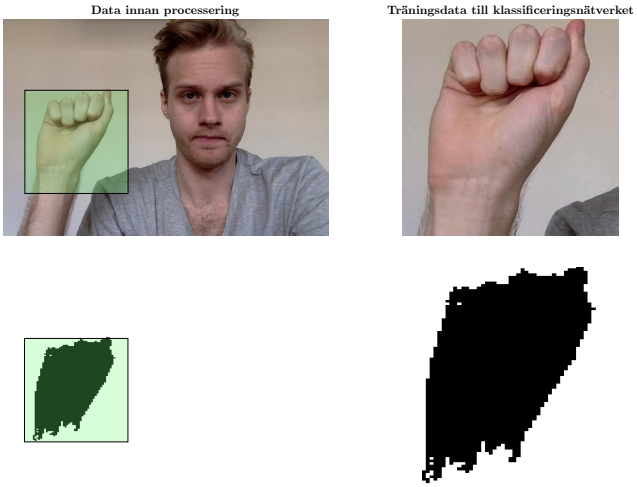
3.1.5.3 Val av segmenteringsnätverk

För att välja vilket av de tränade segmenteringsnätverken som skulle användas för slutprodukten gjordes tester i två etapper. De underpresterande nätverken kunde sorteras ut direkt genom att studera tränings- och valideringsfel samt att rent okulärt studera de masker som producerats av dem. En mer rigorös analys av nätverkens prestanda utfördes med ett beräknat Jaccard-index för ett stort set med bilder som liknade möjliga fall vid faktisk användning. Resultatet av detta visas i Tabell 4.2 samt i Figur 4.5b. De nätverk som har använts i denna studie presenteras tillsammans med relevanta parametrar i Tabell 3.1.

3.1.5.4 Träning av klassificeringsnätverk

Träning av klassificeringsnätverket gjordes med data producerad av segmenteringsnätverket. Med ett väl fungerande segmenteringsnätverk kunde framtagning av masker

[3] Detta till följd av att indatastorleken var $(29 \times 29 \times 3)$.



Figur 3.8: Figuren visar anpassande av bild med hjälp av dess mask. Både mask och bild beskärs beroende på minsta kvadraten kring handen i masken. Notera att även normalisering och LUV-transformation sker inför att datan skall användas till klassificeringsnätverket.

påbörjas för en stor mängd bilder. Datan anpassades genom att förkasta all bildinformation förutom den minsta kvadrat som innehåller handen, genomföra ett byte av färgrymd till LUV och utföra normalisering av de tre kanalerna. Träningsdata till klassificeringsnätverket representerades i fyra kanaler; en normaliserad bild i LUV-färgrymden samt den binära segmenteringsmasken. Hur den ursprungliga bilden beskars visas i Figur 3.8.

Förhoppningen med att bifoga både bild och segmenteringsmask till träning av klassificeringsnätverket var att det enklare skulle kunna utvärdera de tecken som har små variationer sinsemellan. Exempelvis har tecknen för *m, n, s* och *t* skillnader som är svåra att uppfatta genom att endast studera den binära masken, men nätverket hade då data från den ursprungliga bilden att tillgå för att uppfatta dessa variationer. Att bifoga hela bilden ledde även till att nätverket inte blev lika beroende av en perfekt mask.

För att undersöka vilka tecken klassificeringsnätverket hade svårt för producerades en förväxlingsmatris utifrån träningsdata. Den resulterande förväxlingsmatrisen visas i Appendix C som Figur C.1. Det var tydligt att *s* var den bokstav som var svårast att klassificera, antagligen därför att det fanns färre *s* än övriga bokstäver i träningssetet. Som åtgärd valdes att skapa mer träningsdata för bokstaven *s* vilket visade sig förbättra resultatet.

3.1.5.5 Val av klassificeringsnätverk

Tre nätverk tränades för att klassificera tecken. Jämförelse mellan dessa finns presenterade i Tabell 3.2. För att mäta nätverkens prestanda skapades ett dataset med 757 bilder, för vilka masker togs fram genom segmenteringsnätverket *CleganeBowl*. Segmenteringsmaskerna användes sedan tillsammans med tillhörande ursprungsbilder för att utföra tester på de kandiderande klassificeringsnätverken.

Efter att bilderna klassificerats av samtliga nätverk kunde förväxlingsmatriser produ-

Tabell 3.2: De tre kandiderande klassificeringsnätverken. En sammanställning av deras resultat finns presenterade i sektion 4.3.2

Nätverksnamn	Bildstorlek	Learning rate	Batch norm.	Epoker
Gadget	128	0.00001	×	470
Rust	64	0.0001		95
Sherlock	64	0.00001	×	1000

ceras. Det visade sig då att även om nätverken presterade klart bättre än det initiala nätverk som presenterades i Figur C.1 så fanns svagheter vid klassificering av vissa tecken.

Detta löstes genom att sammanställa de tre nätverken till ett större klassificeringssystem, *JudgeJudy*, som väger de tre nätverkens utdata för att göra en gissning som över lag är bättre än de gissningar som de individuella nätverken gör. En förväxlingsmatris skapades för detta system, resultaten finns sammanställda i Tabell 4.4 och visualiseras i Figur 4.8.

3.2 Slutprodukt

Slutprodukten sammanställer processen från det att användaren tar en bild på något tecken till att en tolkad bokstav skrivs ut i användargränssnittet. Användargränssnittet implementerar metoder som syftar till att snabba upp och förbättra kvalitén på segmenteringsnätverkets arbete.

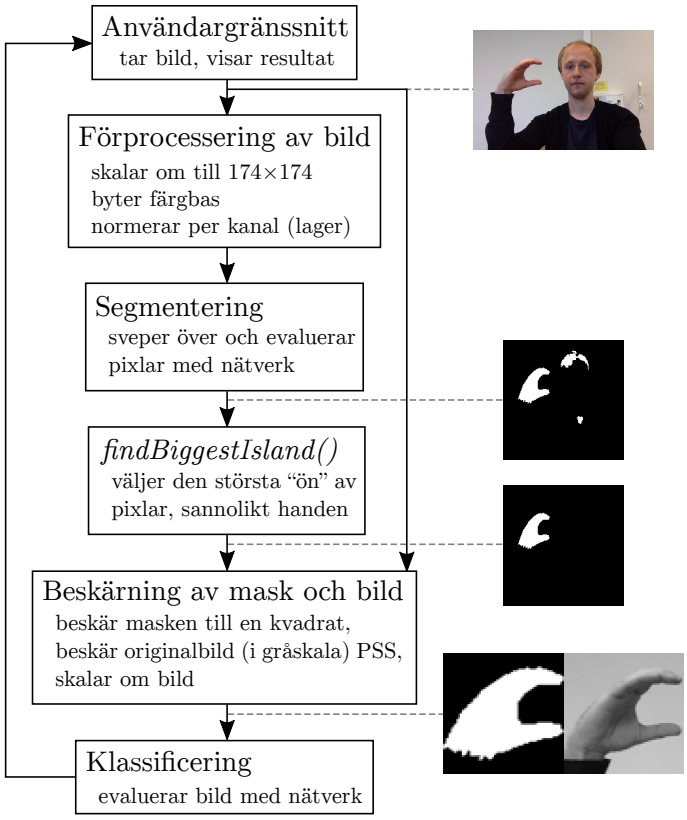
Slutproduktens implementation av ovan beskrivna neurala nätverk och processeringstekniker kan sammanfattas av flödesschemat som visas i Figur 3.9.

3.2.1 Användargränssnitt

Ett användargränssnitt konstruerades med hjälp av Matlabs GUI-skapare GUIDE. Användargränssnittet samlar in bilddata från datorns webbkamera, visar denna i ett fönster, utför evaluering av tagna bilder och visar upp resultatet. Grundstommen i programmet utgörs av en *while*-loop som lämnas då användaren trycker på en knapp varpå den senast tagna bilden skickas till nätverken för evaluering. Efter evalueringen skrivs identifierat tecken ut tillsammans med genererad mask. För att användargränssnittet även skall ge stöd för vänsterhänta finns ett alternativ (en knapp) som gör att indata till evalueringsprocessen speglas, vilket innebär att användaren kan teckna med vänsterhanden. En skärmdump på användargränssnittet finns i Figur 3.10.

3.2.2 Segmentering

Segmenteringsnätverket som evaluerar enstaka pixlar visade sig vara långsamt (storleksordning 1 ms/pixel). För bilder med 174×174 pixlar innebär en naiv svepning



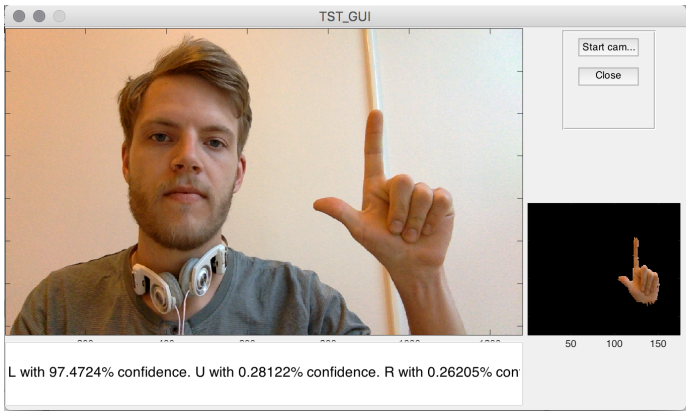
Figur 3.9: Flödesschema som beskriver processen från det att användaren fotograferar en handgest till att en bokstav skrivs ut.

över alla pixlar med ett sådant nätverk en tidsåtgång på omkring 30 s. En smartare svepmetod utvecklades därför för bildsegmenteringen.

3.2.2.1 Smart svepning

Den algoritm som utvecklades reducerar antalet evalueringar av nätverket till enbart antalet pixlar i den hittade masken plus ett litet konstant antal pixlar. Algoritmen bottnar i att inledningsvis grovt evaluera bilden var 2^n:te pixel i varje riktning, vilket innebär att enbart $1/2^{2n}$ av bilden behöver evalueras för att träffa på (tillräckligt stora) objekt. Detta är det lilla konstanta bidraget av evaluerade pixlar för varje bild. De pixlar som nätverket returnerar som träffar läggs i en lista och utgående från denna expanderas sökningen till grannar på ett avstånd 2^{n-1}. Förfarandet upprepas tills inga fler giltiga grannar finns på avstånd 2^{n-1}. Då minskas sökavståndet till 2^{n-2} och processen upprepas. Enkel algoritm/pseudokod visas i Algoritm 1.

Tidskomplexiteten är för det naiva svepet alltid direkt beroende på antalet pixlar i bilden. Det smarta svepet evaluerar (efter en initial grov sökning) enbart områden kring punkter där mask existerar, så den typiska tidsåtgången är betydligt lägre än det naiva svepet (ofta mer än en storleksordning lägre tid).



Figur 3.10: Figuren visar utformningen av det användargränssnitt som utvecklades för körning av algoritmerna.

Algoritm 1 Smart svepning av bild. Används för maskskapande.

Require: $steg \leftarrow 2^n$
Require: *bild* med sidlängd delbar med *steg*
1: *kandidater* ← {alla (x, y) inom *image* där x och y är delbara med *steg*}
2: *träffar* ← {alla k inom *kandidater* där k är del av föremål}
3: **repeat**
4: *steg* ← *steg*/2
5: **repeat**
6: *grannar* ← {alla grannar på avstånd *steg* till något element i *träffar*}
7: *kandidater* ← {alla *grannar* som inte redan evaluerats}
8: *träffar* ← *träffar* + {alla k inom *kandidater* där k är del av föremål}
9: **until** *kandidater* är tom
10: **until** $steg = 1$
11: **return** träffar

3.2.2.2 Tröskelvärden

Då utdatan från segmenteringsnätverket samlas i en matris med sannolikheter används tröskelvärden för att avgöra vilka pixlar som räknas som hand. Tröskelvärdet valdes till $0,3$ vilket medför att samtliga pixlar som nätverket klassificerar som hand med sannolikhet $P > 0,3$ tolkas som hand. Beslutet togs genom att testa olika tröskelvärden och se vilket värde som genererade bäst masker.

3.2.2.3 Metoden findBiggestIsland

Även efter det att tröskelvärdet i svepmetoden optimerats förekom felaktigt klassificerade områden i utdatan från segmenteringsnätverket. För att åtgärda detta skrevs en metod för att extrahera den största sammanhängande komponententen[4] ur den genererade masken. Ett felaktigt segmenterat område svarar typiskt mot ett område i ansiktet på användaren som hamnar i samma ljus som handen. Metoden visade sig

[4]En sammanhängande komponent i en binär mask definieras som ett område där varje pixel i masken har minst en angränsande pixel som också är i masken.

effektiv och används i den färdiga produkten, resultatet av metoden *findBiggestIsland* visas i den sista kolonnen i Figur 4.3.

3.2.3 Klassificering

Som tidigare nämnt i sektion 3.1.5.5 användes inte ett specifikt klassificeringsnätverk för att klassificera utdatan från den totala segmenteringsprocessen. Istället användes en algoritm som viktar ihop gissningar från tre stycken klassificeringsnätverk.

För tre nätverk som ger ut k klasser så definieras en ”poängfunktion” $w = \{2^{(k-n)} - 1 | n = 1, \ldots, k\}$ där poäng w_i ges till den i:te högst sannolika klassen för ett nät. Denna poängfunktion är definierad sådan att om en majoritet av de tre nätverken håller med om en bästa klass, så vinner den klassen alltid (med som lägst en poängs marginal). Andra poängfunktioner kan mycket väl användas (kanske en multiplikativ snarare än summabaserad), men den funktion som användes fungerade tillräckligt bra för att ge en förbättring över de individuella nätverken.

4

Resultat

Ett segmenteringsnätverk och ett klassificeringssystem valdes ut för användning i slutprodukten. I följande avsnitt presenteras utvärdering av slutprodukten, de båda nätverkens prestanda och de resultat som ledde till valen av nätverk.

4.1 Slutprodukten

De två system som implementerades i slutprodukten var segmenteringsnätverket *CleganeBowl* och klassificeringssystemet *JudgeJudy*. Resultaten som presenteras i denna sektion syftar till att utvärdera slutprodukten som helhet.

4.1.1 Prestanda

En förväxlingsmatris som ger ett mått på hur väl den färdiga produkten arbetar togs fram och presenteras i Figur 4.1. Förväxlingsmatriser för de övriga två klassificeringsnätverken kan ses i Figur C.1 i appendix.

4.1.2 Evalueringstid

Evalueringstider[1] för processen som implementeras av användargränssnittet presenteras i Figur 4.2. Medelvärden av tidsåtgång för de olika delarna av evalueringsprocessen presenteras i Tabell 4.1.

Tabell 4.1: Tabellen visar medelvärden för tidsåtgång för olika delar av processen vid evaluering av 757 bilder, som ej använts vid träning.

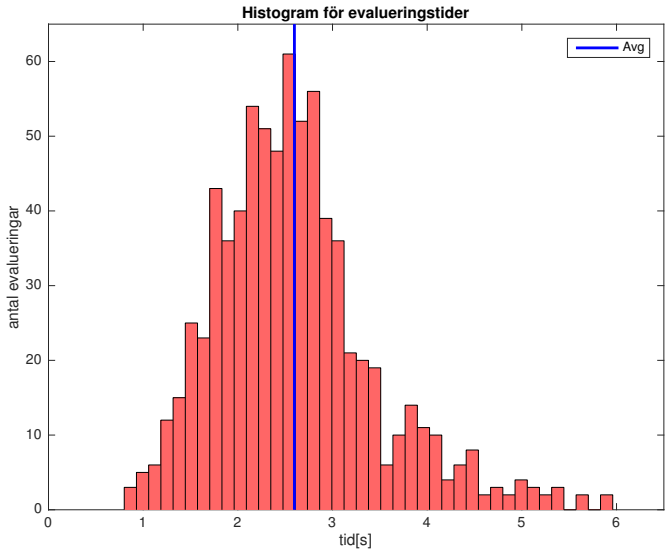
Process	**Medelvärde av evalueringstid**
Förprocessering av bilddata	0,0290 s
Segmentering med *CleganeBowl* via *smart svepning*	2,4801 s
findBiggestIsland samt förprocessering inför klassificering	0,0650 s
Klassificering med *JudgeJudy*	0,0233 s
Total	2,5975 s

[1]Evalueringen skedde på en Macbook Pro med en 2.4 GHz Intel Core i5-processor.

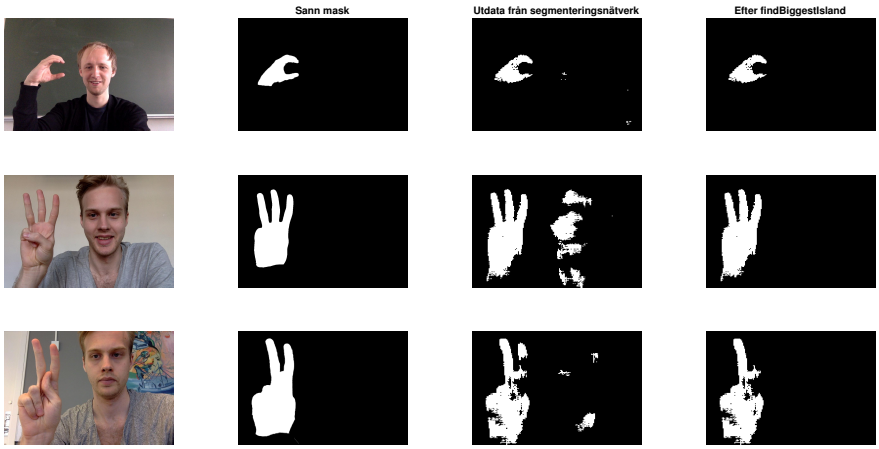
Förutsägelse

Verklighet	A	B	C	D	E	F	G	H	I	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Känslighet
A	19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	6	4	1	0	1	0	0	61 %
B	1	25	0	0	1	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	78 %
C	0	0	30	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	93 %
D	0	0	0	24	0	1	0	1	1	0	1	0	0	0	0	0	1	1	1	0	0	0	1	0	75 %
E	0	2	0	0	26	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	1	81 %
F	0	0	0	0	0	30	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	96 %
G	0	0	0	0	1	0	26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	0	83 %
H	0	0	1	0	0	0	8	20	0	0	0	0	0	1	0	0	0	0	0	0	0	0	2	0	62 %
I	1	0	0	0	0	0	0	0	28	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	93 %
K	0	0	0	0	0	0	0	0	0	25	0	0	1	0	0	0	1	0	0	0	2	3	0	0	78 %
L	0	0	0	0	0	0	0	1	0	0	31	0	0	0	0	0	0	0	0	0	0	0	0	0	96 %
M	0	0	0	0	1	0	0	0	0	0	0	23	2	0	1	0	0	1	0	0	0	0	1	0	79 %
N	0	1	0	0	0	0	0	0	0	0	0	1	27	0	0	0	0	2	0	0	0	0	0	0	87 %
O	0	0	1	1	0	0	0	1	0	0	0	0	0	29	0	0	0	0	0	0	0	0	0	0	90 %
P	1	0	0	0	0	0	0	3	0	0	0	0	0	0	26	2	0	0	0	0	0	0	1	0	78 %
Q	0	0	0	0	0	0	0	2	0	0	1	0	0	0	1	26	1	0	0	0	0	0	1	0	81 %
R	0	1	0	2	1	0	0	0	0	0	0	0	0	0	0	0	22	0	0	3	0	1	2	0	68 %
S	0	0	0	0	6	0	1	0	0	0	0	1	0	0	0	0	0	22	1	0	0	0	0	1	68 %
T	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	28	0	0	0	0	0	87 %
U	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	3	0	0	26	1	0	0	0	81 %
V	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	25	4	0	0	80 %
W	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	30	0	0	96 %
X	0	0	2	1	0	0	1	1	0	0	0	2	1	1	0	0	0	0	0	0	0	0	23	0	71 %
Y	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	30	96 %
Felklassificeringsandel	24 %	16 %	11 %	14 %	27 %	23 %	29 %	31 %	6 %	3 %	6 %	23 %	12 %	6 %	7 %	7 %	21 %	33 %	19 %	13 %	10 %	25 %	34 %	9 %	

Figur 4.1: Förväxlingsmatris för systemet i slutprodukten. Elementet på rad i och kolonn j svarar mot antalet bilder som visar tecknet c_i men som klassificerats som c_j. Hur *känslighet* och *felklassificeringsandel* beräknas finns beskrivna i sektion 2.2.2. Notera att de element som finns utanför diagonalen är förhållandevis starkt grupperade inom $\{a, e, m, n, s, t\}$, $\{g, h\}$, $\{k, v, w\}$ och $\{u, r\}$, de tecken som grupperar sig liknar varandra i ASL, se exempelvis Figur 5.1. Matrisen är genererad genom evaluering av en uppsättning bilder som *inte* använts som träningsdata.



Figur 4.2: Histogrammet visar tidsåtgången per bild för hela evalueringsprocessen för 757 bilder.



Figur 4.3: Exempel på segmentering av bilder med nätverket *CleganeBowl*. Den tredje kolonnen visar masker som genererats av nätverket med tröskelvärdet 0,3. Vi ser i den fjärde kolonnen masker efter att metoden *findBiggestIsland* använts. I nedersta raden finns ett exempel där *findBiggestIsland* skurit bort en del av ett finger eftersom det inte är anknutet till resten av handen.

4.2 Segmentering

I slutskedet av projektet utvärderades prestantan för sex kandiderande segmenteringsnät, se Tabell 3.1. Det som skiljer dessa nätverk åt är vilken data de tränats med samt vilka metoder och parametrar som använts vid träning. Avsnittet kommer fokusera på resultat från nätverket *CleganeBowl* men syftar även till att jämföra de olika segmenteringsnätverkens prestanda. I Figur 4.3 visas några exempel på masker från segmenteringsnätverket *CleganeBowl*.

4.2.1 Tränings- och valideringsfel

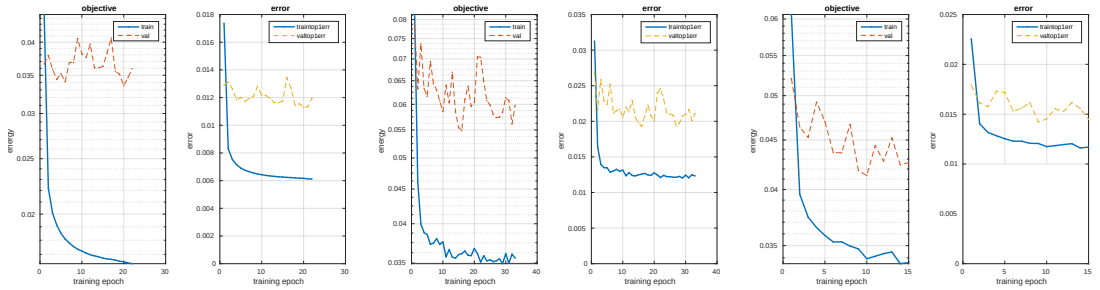
Under träning mättes både tränings- och valideringsfelet för samtliga nät. Dessa fel finns beskrivna för de sex nätverken i Figur 4.4.

4.2.2 Utdata från olika lager

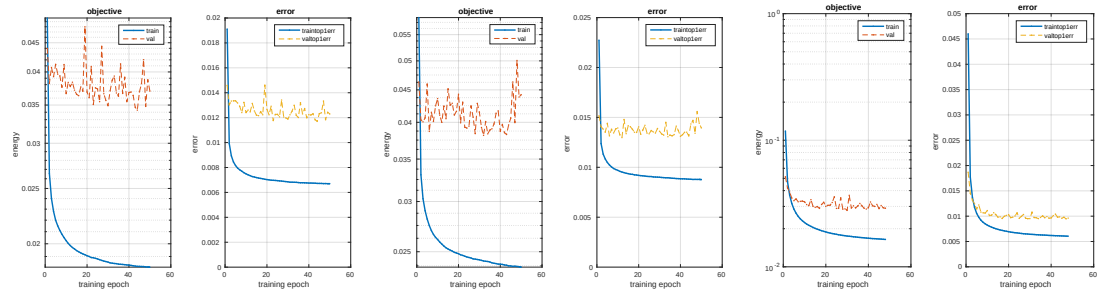
För att få en uppfattning av vad som händer med en bild vid evaluering i segmenteringsnätverket har data hämtats från olika delar av processen. Denna data presenteras i Figur 4.7.

4.2.3 Jaccard-index

Jaccard-index för de sex olika segmenteringsnätverken beräknades från ett dataset med 144 bilder och presenteras i Figur 4.5. Medelvärdet av dessa index kan ses i Tabell 4.2. Indexen är beräknade både före och efter det att *findBiggestIsland* använts (se sektion 3.2.2.3) och ger ett mått på hur väl segmenteringen utförs med och utan hjälpfunktioner.

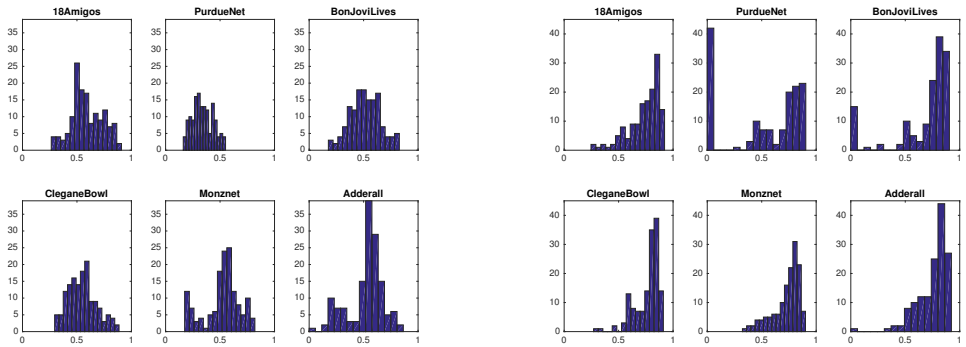


(a) PurdueNet (b) 18Amigos (c) BonJoviLives



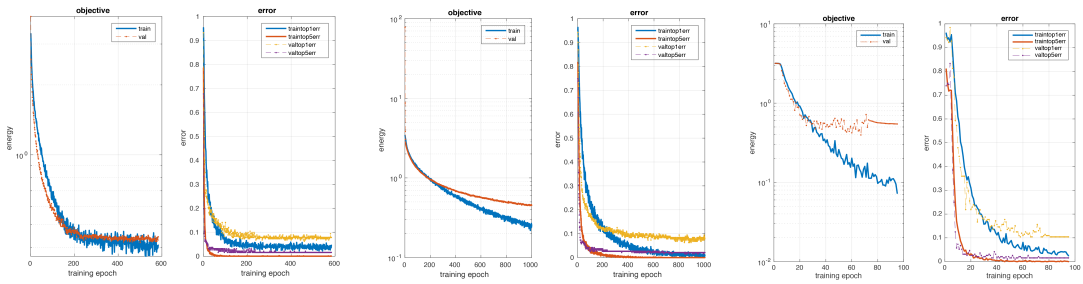
(d) Monznet (e) Adderall (f) CleganeBowl

Figur 4.4: Figurerna visar hur felfunktion och *top1-error* förändras vid träning av de olika segementeringsnätverken. *top1-error* beskriver hur ofta nätverkets gissning inte överensstämmer med sanningen.



(a) Före användning av *findBiggestIsland*. (b) Efter användning av *findBiggestIsland*.

Figur 4.5: Histogram över Jaccard-index för de sex segmenteringsnätverken före och efter användning av *findBiggestIsland*.

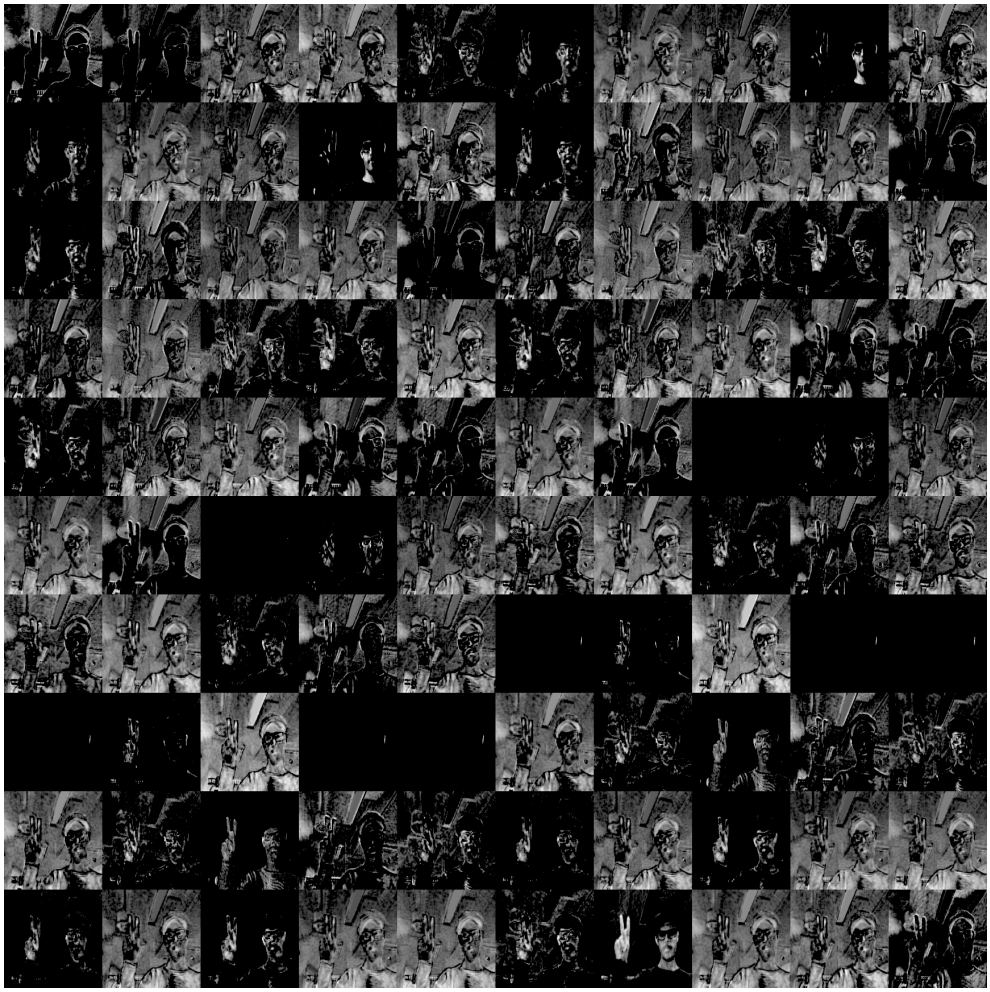


(a) Gadget (b) Sherlock (c) Rust

Figur 4.6: Figurerna visar hur felfunktionen, *top1-error* och *top5-error* förändras vid träning av de tre olika klassificeringsnätverken. *top5-error* beskriver hur ofta ingen av nätverkets fem bästa gissningar överensstämmer med sanningen.

(a) Utdata efter första blocket (faltning, pooling och aktivering).

(b) Utdata efter andra blocket (faltning, pooling och aktivering).



(c) Utdata efter tredje faltnings- och aktiveringslagret.

Figur 4.7: Figurerna visar utdatan efter några steg av evaluering med segmenteringsnätverket. För att göra mörka partier tydligare visas kvadratroten av de faktiska värdena. Notera hur olika grenar (bilder) specialiserar sig på olika saker. En del förtydligar handen, en del exkluderar bakgrund, en del exkluderar ansikte och så vidare. Alla grenarna kommer kombineras av fullt kopplade lager som idealt plockar de bästa delarna från varje gren för att skapa ett särdrag (en *feature*).

Tabell 4.3: Skillnader i tidsåtgång vid segmentering beroende på smart svepning.

Process	Medelvärde av evalueringstid	Genomsnittlig andel utvärderade pixlar
Segmentering med *smart svepning*	2,4801 s	8,602%
Segmentering utan *smart svepning*	29 s	100%

Tabell 4.2: Medelvärden av Jaccard-index för de sex segmenteringsnätverken. Vänster data-kolonn är beräknad direkt efter segmentering, höger data-kolonn är beräknad för samma data efter det att metoden *findBiggestIsland* använts.

Nätverksnamn	Medelvärde från segmentering	Medelvärde med findBiggestIsland
18Amigos	0,5909	0,7367
PurdueNet	0,3418	0,5042
BonJoviLives	0,5117	0,6783
CleganeBowl	0,5492	0,7635
Monznet	0,5217	0,7274
Adderall	0,5201	0,7523

4.2.4 Tidsvinst från smart svepning

Tidsvinsten från metoden *smart svepning* (se sektion 3.2.2.1) uppskattas genom att evaluera ett stort antal bilder och spara procentandelen av den totala mängden pixlar som har evaluerats. Tiden det hade tagit utan smart svepning kan då estimeras. Denna uppskattning presenteras i Tabell 4.3.

4.3 Klassificering

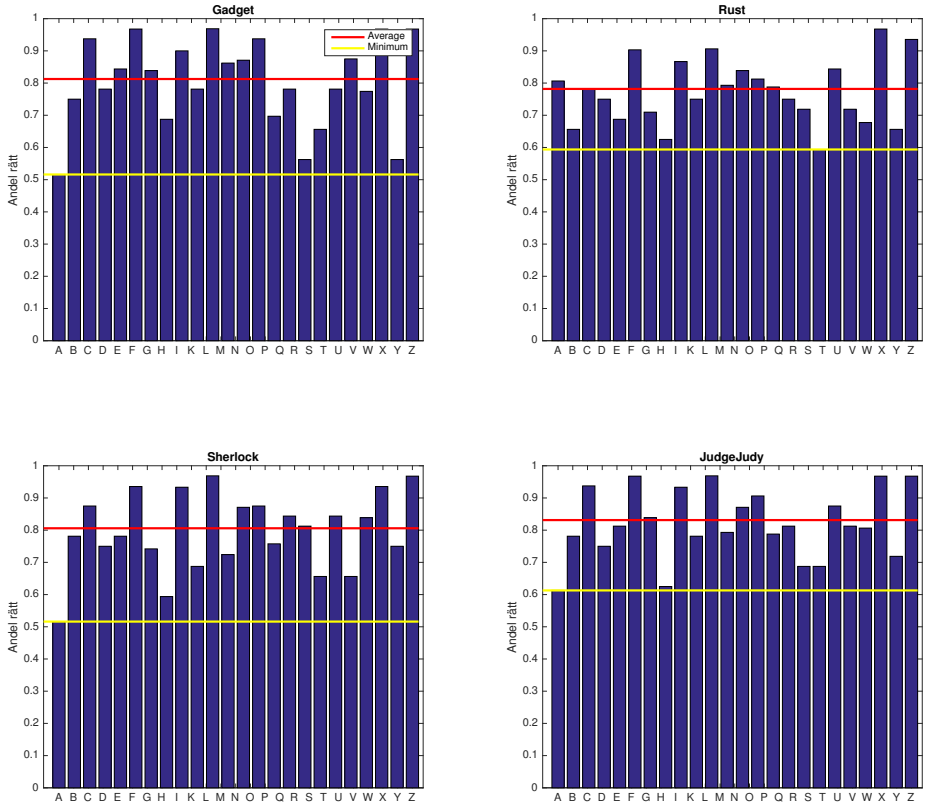
I denna sektion presenteras resultat från de tre klassificeringsnätverkens träning, jämförelser av nätverkens och klassificeringssystemet *JudgeJudys* precision och exempel på hur ett klassificeringsnätverk kan manipulera bilddata.

4.3.1 Tränings- och valideringsfel

Under träning av de tre klassificeringsnätverken mättes både tränings- och valideringsfelet. Hur felen minskar med ökad träning visas i Figur 4.6.

4.3.2 Sammanställning av förväxlingsmatriser

Förväxlingsmatriser togs fram för de tre nätverken och klassificeringssystemet. Relevant data extraherades ur testresultat från dessa och finns presenterade i Figur 4.8. Exakta värden för nätverkens genomsnittliga och lägsta andel korrekta gissningar presenteras i Tabell 4.4.



Figur 4.8: Histogrammet visar hur väl de olika nätverken klassificerade bilder ur ett bildset med 757 bilder som ej använts som träningsdata. De blå staplarna visar andelen gissningar som var korrekta för respektive bokstav. Den röda linjen svarar mot den genomsnittliga andelen korrekta gissningar och den gula linjen den lägsta andel korrekta gissningar för någon bokstav. Exakt data för dessa två värden presenteras i Tabell 4.4.

Tabell 4.4: Exakta värden för genomsnittlig andel korrekta gissningar och lägsta andel för korrekta gissningar visualiserade i Figur 4.8.

Nätverksnamn	**Genomsnittlig andel korrekta gissningar**	**Lägsta andel för korrekta gissningar**
Gadget	0,8126	0,5161
Rust	0,7818	0,5838
Sherlock	0,8059	0,5161
JudgeJudy	0,8313	0,6129

5

Diskussion

Ur det resultat som presenterats ser vi att den slutprodukt som implementerar vår process, med vissa avgränsningar och under goda förhållanden, kan tolka teckenspråk. I följande avsnitt kommer en diskussion föras kring de resultat som tagits fram i ett försök att förstå skillnader mellan de förväntade resultaten och de observerade. Vi kommer även att diskutera hur arbetet hade kunnat fortsätta om projektet hade förlängts.

5.1 Segmentering

Det var en lång process att ta fram ett segmenteringsnätverk som fungerar väl. Att vi lyckats använda nätverket *CleganeBowl* för att producera träningsdata till klassificeringsnätverk visar att segmenteringsnätverket kan ge goda resultat. Däremot finns förhållanden då segmenteringsnätverket underpresterar, exempelvis blir resultaten dåliga vid vissa ljussättningar. Segmenteringen står också för en överväldigande majoritet av den totala evalueringstiden vilket inte är önskvärt. Genom att analysera resultaten för segmentering hoppas vi förstå vad dessa brister beror på.

5.1.1 Förbehandlingens betydelse

Segmentering fungerar bra för bilder tagna inomhus i indirekt ljus och med någorlunda homogen bakgrund. Segmentering av bilder tagna i motljus eller som har ljuskällor med i bakgrunden misslyckas ofta, problematik uppstår också om bakgrunden är av ungefär samma färg som hudfärg eller ifall den är allmänt rörig. I motsats till vad vi trodde i början av projektet så lyckades vi skapa ett segmenteringsnätverk som är bra på att segmentera bilder utan att ansiktet systematiskt tolkas som hand. I de fall där ansiktet ändå tolkas som hand, som i Figur 4.3, så kan metoden *findBiggestIsland* ofta plocka bort ansiktet.

Från resultaten är det svårt att utröna om vår förbehandling av bilddatan är viktig, något som hade testats om tid funnits. Normalisering och LUV-transform lades till med förhoppningen att ljusstyrka och färg i bilderna skulle kunna homogeniseras. Vinningen med metoderna kan dock ifrågasättas då vi i slutprodukten fortfarande har problem med bilder tagna i extrema ljusförhållanden.

5.1.2 Bildmaskernas kvalitet

De sex olika segmenteringsnätverken som presenteras i Tabell 4.2 och Figur 4.5 visade sig producera masker med olika typer av fel. Vi valde att fortsätta utveckla produkten för de nätverk som var mer generösa, det vill säga tolkade för mycket som hand snarare än tvärt om. Detta gjordes för att metoden *findBiggestIsland* visade sig kraftfull och ofta bevarar en korrekt mask av handen. *CleganeBowl* är ett nätverk där utdatan oftast är en tydlig hand och en liten mängd brus, vilket enkelt åtgärdas *findBiggestIsland*.

Segmenteringsnätverken har generellt sett svårt att dra gränsen mellan hand och arm. När träningsdata producerades för segmentering drogs gränsen vid handleden, men utdatan från segmenteringsnätverken visar ofta en bit av underarmen.

5.1.3 Tidsåtgång för segmentering

Som vi ser i Tabell 4.1 utgör segmenteringen merparten av den totala evalueringstiden trots användandet av *smart svepning*. Detta beror på den pixelvisa segmenteringen, alltså att varje pixel evalueras separat. Trots att vi endast evaluerar i genomsnitt 8.6% av bilden, via *smart svepning*, krävs ca 2600 evalueringar av nätverket. Eftersom vi evaluerar varje pixel tillsammans med en (29×29) pixlar stor omgivning finns ett överlapp som gör att varje pixel tas i beaktning upprepade gånger. Om vi minskar området som beaktas vid evaluering minskar överlappet och vi får snabbare evaluering men då får också segmenteringsnätverken mindre information att arbeta med vilket torde öka felaktigt klassificerade pixlar.

Ett annat alternativ för att sänka tidsåtgången för segmentering vore att extrahera viktmatriserna så att en hel bild kan behandlas direkt, vilket hade förkortat segmenteringstiden med flera storleksordningar. Extrahering av viktmatriserna beskrivs mer noggrant i sektion 5.3.1.

5.2 Klassificering

Ett generellt resultat för klassiceringsnätverken erhölls ur valideringsfelet samt andelen korrekta gissningar för testdata-setet. Som vi ser i Figur 4.6 så hamnar valideringsfelet under träning på ungefär 10%. Om detta jämförs med de genomsnittliga andelarna korrekta gissningar i Tabell 4.4 ser vi att det finns en viss diskrepans. Diskrepansen beror på skillnaden mellan valideringsdata och testdata; valideringsdatan insamlades tillsammans med träningsdatan och när dessa producerades sållades masker av lägre kvalitet ut. Detsamma gjordes ej för testdatan vilket leder till att klassificeringsnätverken då får indata som är svårare att identifiera.

5.2.1 Förväxling av tecken

Som Figur 4.1 visar förväxlar klassificeringsnätverken vissa tecken i högre grad än andra. Exempelvis förväxlas m med n i viss utsträckning, samt h med g och k med v.

(a) *g* och *h* **(b)** *k* och *v* **(c)** *m* och *n*

Figur 5.1: Figurerna visar några av de grupper tecken som nätverket tenderar att förväxla. Notera att tecknen i grupperna är lika varandra till formen.

Figur 5.1 visar att dessa bokstäver tecknas liknande och detta kan ge en förklaring till varför de är svåra att särskilja.

5.2.2 Behovet av tre nätverk

Som vi kan se i Figur 4.8 och Tabell 4.4 så ger klassificeringssystemet ett bättre resultat än något av de andra klassificeringsnätverken individuellt. Idealt hade vi velat ha ett nätverk som utför sin uppgift minst lika bra som klassificeringssystemet, men praktiskt sett så lider inte slutprodukten speciellt mycket av att tre klassificeringsnätverk används då klassificeringstiden är mycket kort jämfört med segmenteringstiden. Vid användning av ett klassificeringssystem kan vi inte få en ordentlig kvantifiering av säkerheten på en gissning. Vi anser trots detta att klassificeringssystemet är en förbättring av produkten då andelen felklassificerade tecken minskar.

Idag viktas gissningarna från de olika nätverken lika högt men från Figur 4.8 ser vi att nätverket *Rust* identifierar bokstaven *a* bättre än de andra nätverken, både i termer av känslighet och felklassificeringsandel (förväxlingsmatris finns i appendix C). Om man hade velat fortsätta utveckla klassificeringssystemet så kan algoritmen som viktar de olika nätverkens gissningar förbättras så att nätverkens resultat viktas olika beroende på tecken. Detta hade medfört att klassificeringssystemet i högre grad hade litat på gissningar från nätverk som uppvisat hög säkerhet på ett visst tecken. Denna säkerhet skulle kunna beräknas utifrån de känsligheter och felklassificeringsandelar som kan ses i förväxlingsmatriserna.

5.3 Vidareutveckling av produkten

Det finns flera möjliga vidareutvecklingar av produkten som inte implementerats till följd av den begränsade tiden för projektet. En del av dessa vidareutvecklingar har resonerats kring inom arbetsgruppen och lösningar för att implementera dem diskuteras i denna sektion.

5.3.1 Snabbare evaluering

Kortare evalueringstid hade öppnat för kontinuerlig utvärdering av bilddata. Försök gjordes under arbetets gång att hitta metoder för att kunna tillämpa det färdigtränade pixelnätverket på hela bilder i en körning. Tidsbrist gjorde att ingen färdig metod konstruerades innan primär deadline, men det konstaterades att evalueringstiden skulle kunna minskas drastiskt (storleksordningar) och ett exempel visades i Figur 5.2.

Figur 5.2: Mask segmenterad genom matrismässig evaluering av en hel bild. Bilden blir mindre på grund av pooling-lager, men det är möjligt att jobba runt detta problem på smarta sätt så att resultatet är en fullstor bild.

Exemplet visar en grov bild, men det anmärktes att det är möjligt att få ut en fullstor bild. Efter primär deadline färdigställdes en sådan metod för vilken en redogörelse kan ses i Appendix E. Då metoden färdigställdes efter deadline kommer rapporten inte skrivas om med detta resultat i åtanke (undantaget denna sektion).

5.3.2 Mer träningsdata

För att förbättra resultatet för den slutgiltiga produkten behövs antagligen mer träningsdata till klassificeringsnätveket. För träning av klassificeringsnätverket användes ungefär 750 bilder vilket motsvarar ungefär 32 bilder per klass, detta är en väldigt liten databas ifall den jämförs med exempelvis segmenteringsnätet som, om 8% av varje bild är hand, tränas med 2400 bilder för ena klassen och 28000 bilder för den andra för varje enskild hel bild (174×174 pixlar). Dessa tal multipliceras alltså med antalet hela bilder i träningsdatan för att få de verkliga jämförelsetalen. För *CleganeBowl* som tränades med en databas om ungefär 3000 bilder ger detta 7.2×10^6 bilder för ena klassen och 85×10^6 för den andra.

Vi tror inte att miljontals bilder behövs för att klassificeringsnätet ska lära sig att säkert skilja på olika tecken och vill förtydliga att olika stora träningsdataset krävs för olika nätverk, men att använda enbart 32 bilder per klass för träning kan vara aningen lågt om man strävar efter en säker klassificering.

5.3.3 Utökning av teckenvokabulär

En naturlig vidareutveckling av programmet är utökning av dess teckenvokabulär. Det är oklart hur en sådan vidareutveckling skulle genomföras då det är svårt att välja in en ny grupp av tecken, utan att förvirra användaren då det blir oklart vilka tecken som är möjliga att göra och vilka som inte är det. Problem i samma avseende är den stora mängd ickestatiska tecken som finns i ASL, samt tecken som kräver användning av både händerna. Vissa tecken tecknas framför ansiktet på användaren vilket skapar ytterligare problem för befintliga nät.

Näten måste, för att klara av nya typer av tecken, byggas om till följd av att stor vikt lagts vid att programmet ska klara av att segmentera bort ansikten och övrig hud samt arbeta på statiska tecken med endast en hand. Det bör här poängteras att alla statiska tecken som kan göras med en hand och som inte tecknas framför

ansiktet enkelt borde kunna läggas till, men att det krävs en större mängd bilder, med tillhörande masker, per tecken för att träna detta.

5.3.3.1 Tecknande med båda händerna

För att hantera tecken som görs med båda händerna skulle segmenteringsnätverket kunna tränas för att känna igen vilken hand som är högerhand respektive vänsterhand. Huruvida detta går att genomföra med befintligt nätverk är oklart och andra lösningar kan eventuellt finnas. För att hantera ickestatiska tecken som tecknas med båda händerna behöver produkten antagligen revideras kraftigt.

5.3.3.2 Ickestatiska tecken

Vidareutveckling av produkten för att möjliggöra hantering av tecken som kräver en handrörelse hade lagt en grund för ett mycket större teckenvokabulär.

För att kunna tolka ett dynamiskt tecken krävs att evalueringstiden är tillräckligt kort för att kunna fånga rörelsen i tecknet. Eftersom evaluering av bilder i nuläget tar lång tid omöjliggörs denna vidareutveckling med befintliga nätverk. Ifall ett snabbare segmenteringsnätverk utvecklas hade ett alternativ varit att analysera en ny bild direkt efter det att den tidigare bilden blivit färdig. Detta hade lett till att teckenföljder hade kunnat översättas till ickestatiska tecken.

Denna typen av vidareutveckling går till sin natur ifrån vad projektet omfattar och en helt ny typ av datahantering måste utvecklas.

5.3.3.3 Tecknande framför ansiktet

För att hantera tecken som görs framför ansiktet på användaren kan segmenteringsnätveket utökas för att hantera en tredje klass. Idag finns klasserna *hand* och *bakgrund* implementerade, men nätet bör kunna byggas ut för att även hantera klassen *ansikte*. Om detta görs borde klassificeringsnätet kunna tränas för att hantera tecken som placeras framför ansiktet.

5.3.4 Byte av programmeringsspråk

För att öka tillgängligheten av den utvecklade applikationen bör koden översättas till ett öppet programmeringsspråk. Då koden ursprungligen är skriven i Matlab som är både nischat och dyrt i inköp, hade ett bättre val kanske varit att använda Java, Python eller C/C++. Byte till ett lämpligare programmeringsspråk hade underlättat distribution av programmet.

Att utföra en sådan portering bör vara förhållandevis enkelt att genomföra. De färdigtränade nätverken består i grunden av enklare viktmatriser och endast den funktionalitet som erhålls av varje lager behöver implementeras för att använda dem. Med detta gjort hade samma funktionalitet erhållits för nya plattformar.

6

Slutsatser

Trots att vi riktat kritik mot prestandan för de olika delarna i evalueringsprocessen så är vi i stort nöjda med slutprodukten. För en tillräckligt bra miljö så kan teckenspråkstolken klassificera rätt tecken ur alfabetet i genomsnitt 83% av gångerna. Tack vare den tidsvinst som fås via metoden *smart svepning* anser vi att tiden det tar att evaluera en bild är tillräckligt kort för att produkten ska kännas användbar. Produkten i sin nuvarande form hade antagligen inte blivit en jättestor tillgång i kommunikation mellan teckenspråksanvändare och icke teckenspråksanvändare men däremot skulle den kunna anpassas till att bli ett verktyg för utlärning av teckenspråk.

Vid användande av neurala nätverk har vi märkt vilket kraftigt verktyg det kan vara, trots att vi bara skrapat på ytan av dess tillämpningsområden. En intressant observation är att vi under arbetet märkt att vi vill lösa allt fler problem med hjälp av neurala nätverk. Det hade exempelvis varit intressant att undersöka om algoritmen som viktar gissningar i klassificeringssystemet kan ersättas med ett neuralt nätverk.

Den naturliga fortsättningen för oss, tillika vår rekommendation för en efterkommande projektgrupp, är att försöka använda ett segmenteringsnätverk som använder matrismässig evaluering istället för elementvis (pixelvis) evaluering, då den senare typen har visat sig vara långsam.

7

Bibliografi

[1] D. Hebb. *The Organization of Behavior*. McGill University, New York, 1949.

[2] W. S. McCulloch & W. Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[3] B. G. Farley & W. Clark. Simulation of self-organizing systems by digital computer. *Information Theory, Transactions of the IRE Professional Group on*, 4(4):76–84, September 1954.

[4] P. J. Werbos. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. PhD thesis, Harvard University, 1974. Department of Applied Mathematics.

[5] IBM Research. Deep Blue. `http://www.research.ibm.com/deepblue/`, 1997.

[6] In a Huge Breakthrough, Google's AI beats a Top Player at the Game of Go. `http://www.wired.com/2016/01/in-a-huge-breakthrough-googles-ai-beats-a-top-player-at-the-game-of-go/`. Hämtat: 2016-02-10.

[7] A. Vedaldi & K. Lenc. MatConvNet – Convolutional Neural Networks for MATLAB. In *Proceeding of the ACM Int. Conf. on Multimedia*, 2015.

[8] R. S. Sutton & A. G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, Cambridge, Massachusetts, 2005.

[9] S. Ioffe & C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. `http://arxiv.org/pdf/1502.03167v3.pdf`. Hämtat: 2016-05-17.

[10] Introduction to Colour Spaces, CIE XYZ. `http://www.colourphil.co.uk/xyz_colour_space.shtml`. Hämtat: 2016-04-15.

[11] R. B. Wilbur & A. C. Kak. Purdue RVL-SLLL American Sign Language Database. Technical report, School of Electrical and Computer Engineering, Purdue University, W. Lafayette, IN 47906, 2006. TR-06-12.

[12] M. Kawulok, J. Kawulok & J. Nalepa. Spatial-based skin detection using discriminative skin-presence features. doi: 10.1016/j.patrec.2013.08.028. Hämtat: 2016-05-14.

[13] M. Kawulok. Fast propagation-based skin regions segmentation in color images. doi: 10.1109/FG.2013.6553733, April 2013. Hämtat: 2016-05-14.

[14] T. Grzejszczak & M. Kawulok J. Nalepa. Wrist localization in color images for hand gesture recognition. doi: 10.1007/978-3-319-02309-0_8". Hämtat: 2016-05-14.

[15] M. Kawulok. Database for hand gesture recognition. http://sun.aei.polsl.pl/~mkawulok/gestures/. Hämtat: 2016-05-14.

[16] Matlab File Exchange: Colorspace Transformations. http://www.mathworks.com/matlabcentral/fileexchange/28790-colorspace-transformations. Hämtat: 2016-04-18.

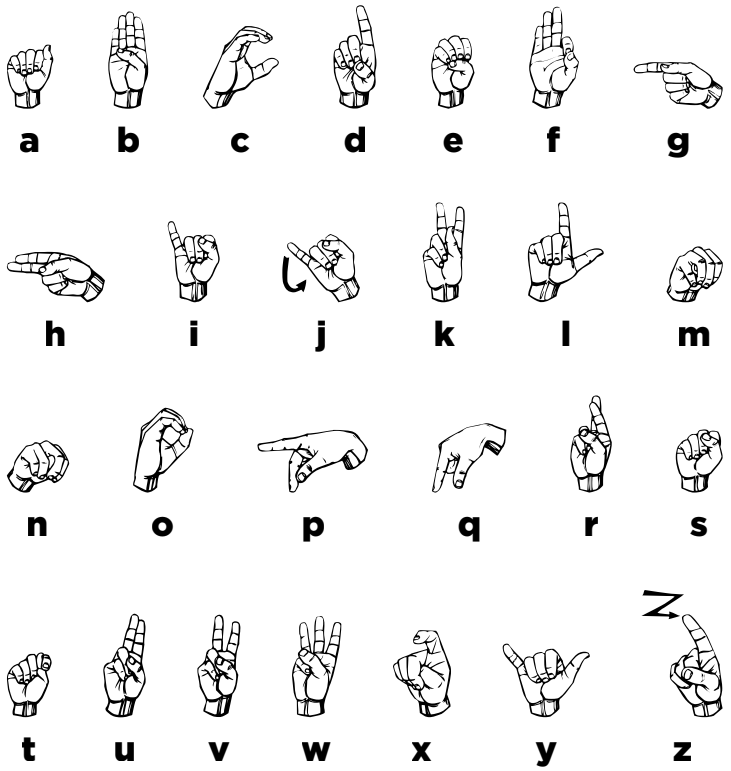
[17] A. Vedaldi & B. Fulkerson. VLFeat: An Open and Portable Library of Computer Vision Algorithms. http://www.vlfeat.org/, 2008.

[18] J. Keane. ASL Fingerspelling Chart. http://jonkeane.com/Asl_alphabet_gallaudet.pdf. Hämtat: 2016-05-12.

A

ASL

Figur A.1 visar alla bokstäver i det amerikanska teckenspråksalfabetet ASL.



Figur A.1: En tabell över bokstäver i amerikanska teckenspråket, ASL. Illustration från Jonathan Keane [18] under Creative Commons Attribution-ShareAlike 3.0 Unported License.

B

Matematik

B.1 Faltning

Faltning (*convolution*) är en binär operator som agerar på två funktioner f och g. Operationen definieras enligt

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)\, g(t-\tau)\, \mathrm{d}\tau$$

och motsvarar geometriskt arean av produkten mellan f och g när g förskjuts med ett avstånd t. Faltning har stora tillämpningar då operationen naturligt dyker upp i samband med Fourier- och Laplacetransform och därigenom också i tillämpningen filter.

B.1.1 Diskret faltning

För att kunna tillämpa faltning i datorsammanhang vill man definiera faltning på diskret form. Omskrivningen är naturlig, och faltningen för två följder f_n och g_n definieras som

$$(f * g)_n = \sum_{m=-\infty}^{\infty} f_m\, g_{n-m}.$$

B.1.2 Flerdimensionell faltning

Faltning generaliseras enkelt till N variabler. För koncishet används här en vektornotation som egentligen motsvarar N integraler eller summor (en per variabel). En N-dimensionell faltning mellan f och g betecknas ofta $f * \overset{N}{\cdots} * g$ och definieras av

$$(f * \overset{N}{\cdots} * g)(\boldsymbol{t}) = \int f(\boldsymbol{\tau})\, g(\boldsymbol{t}-\boldsymbol{\tau})\, \mathrm{d}\boldsymbol{\tau},$$

eller i det diskreta fallet

$$(f * \overset{N}{\cdots} * g)_{\boldsymbol{n}} = \sum_{\boldsymbol{m}} f_{\boldsymbol{m}}\, g_{\boldsymbol{n}-\boldsymbol{m}}.$$

C

Förväxlingsmatriser

Förutsägelse

Verklighet	A	B	C	D	E	F	G	H	I	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Känslighet
A	34	0	0	0	0	0	0	0	0	0	0	8	0	0	0	0	0	0	1	0	0	0	0	0	79 %
B	0	40	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	95 %
C	0	0	37	0	0	0	0	0	0	0	0	1	0	2	0	0	0	0	0	0	0	0	0	0	92 %
D	0	0	0	33	0	0	0	0	1	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	89 %
E	0	1	0	0	32	0	0	0	0	0	0	4	0	0	0	0	0	0	0	0	0	0	0	0	86 %
F	0	1	0	0	0	38	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	97 %
G	0	0	0	0	0	0	31	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	1	1	88 %
H	0	0	0	0	0	0	10	27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	72 %
I	0	0	0	0	0	0	1	0	41	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	97 %
K	0	0	0	1	0	0	0	0	0	35	0	0	0	0	0	0	0	0	0	0	7	2	0	0	77 %
L	0	0	0	1	0	0	0	0	0	0	42	0	0	0	0	0	0	0	0	0	0	0	0	0	97 %
M	1	0	0	0	1	0	0	0	0	0	0	38	0	0	0	0	0	0	0	0	0	0	0	0	95 %
N	2	0	0	0	0	0	0	0	0	0	0	7	29	1	0	1	0	0	0	0	0	0	0	0	72 %
O	0	0	0	0	0	0	0	0	0	0	0	1	0	38	0	0	0	0	0	0	0	2	0	0	92 %
P	0	0	0	0	0	0	0	0	0	0	0	0	0	1	39	0	0	0	0	0	0	0	0	0	97 %
Q	0	0	0	0	0	0	0	0	0	0	0	0	0	1	8	26	0	0	0	0	0	0	0	0	74 %
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	38	0	0	1	0	0	0	0	97 %
S	0	1	0	0	1	0	0	0	0	0	0	7	0	0	0	0	0	0	1	0	0	0	0	0	0 %
T	8	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	33	0	0	0	0	0	75 %
U	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	43	0	0	0	0	95 %
V	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	34	2	0	0	87 %
W	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	35	0	1	92 %
X	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	3	0	0	0	0	0	35	0	89 %
Y	1	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	39	92 %
Felklassificeringsandel	26 %	6 %	0 %	8 %	5 %	0 %	26 %	0 %	2 %	5 %	2 %	47 %	0 %	11 %	20 %	3 %	19 %		5 %	4 %	17 %	14 %	2 %	4 %	

Figur C.1: Förväxlingsmatris för klassificeringsnätverk där elementet på rad i och kolonn j svarar mot antalet bilder som visar tecknet c_i men som klassificerats som c_j. Hur *känslighet* och *felklassificeringsandel* beräknas finns beskrivna i sektion 2.2.2. Notera att de element som finns utanför diagonalen är förhållandevis starkt grupperade inom $\{a, e, m, n, s, t\}$, $\{g, h\}$, $\{k, v, w\}$ och $\{u, r\}$, de tecken som grupperar sig liknar varandra i ASL, se exempelvis Figur 5.1. Matrisen är genererad med träningsbilder och visar hur en förväxlingsmatris kan ge information om hur väl träningen fungerar. I detta fall visade sig datan för s vara bristfällig och ibland felaktig.

Förutsägelse

Verklighet \ Förutsägelse	A	B	C	D	E	F	G	H	I	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	
A	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	6	6	0	0	1	0	2	51 %
B	1	24	0	0	2	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	75 %
C	0	0	30	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	93 %
D	0	0	0	25	0	1	0	0	1	0	1	0	1	0	0	0	1	1	0	0	0	0	1	0	78 %
E	0	1	0	0	27	0	0	0	0	0	0	2	0	1	0	0	0	0	0	0	0	0	0	1	84 %
F	0	0	0	0	0	30	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	96 %
G	0	0	0	0	0	0	26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5	0	83 %
H	0	0	1	0	0	0	6	22	0	0	0	0	0	1	0	0	0	0	0	0	0	0	2	0	68 %
I	1	0	0	0	0	0	0	0	27	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	90 %
K	0	1	1	0	0	0	0	0	0	25	0	0	1	0	0	0	1	0	0	0	2	1	0	0	78 %
L	0	0	0	0	0	0	0	1	0	0	31	0	0	0	0	0	0	0	0	0	0	0	0	0	96 %
M	0	0	0	0	2	0	0	0	0	0	0	25	1	0	1	0	0	0	0	0	0	0	0	0	86 %
N	0	1	0	0	1	0	0	0	0	0	0	1	27	0	0	0	0	1	0	0	0	0	0	0	87 %
O	0	0	1	0	0	0	0	1	0	0	0	0	0	30	0	0	0	0	0	0	0	0	0	0	93 %
P	1	0	0	0	1	0	0	4	0	0	0	1	0	0	23	3	0	0	0	0	0	0	0	0	69 %
Q	0	0	0	0	0	0	0	2	0	0	0	0	0	0	2	25	2	0	0	0	0	0	1	0	78 %
R	0	0	0	4	1	0	0	0	1	0	0	0	0	0	0	0	18	0	0	3	0	1	4	0	56 %
S	1	0	0	0	6	0	0	0	0	0	0	0	0	0	0	0	0	21	1	0	0	0	0	3	65 %
T	4	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	25	0	0	0	0	0	78 %
U	0	0	0	2	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	28	0	0	0	0	87 %
V	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	24	6	0	0	77 %
W	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	30	0	0	96 %
X	0	0	0	4	0	0	3	1	1	0	0	1	1	2	0	0	0	0	1	0	0	0	18	0	56 %
Y	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	30	96 %
Felklassificeringsandel	33 %	14 %	14 %	28 %	32 %	18 %	25 %	31 %	12 %	0 %	3 %	16 %	12 %	16 %	11 %	10 %	21 %	30 %	28 %	9 %	7 %	23 %	41 %	18 %	

Känslighet

Figur C.2: Förväxlingsmatris för klassificeringsnätverket *Gadget*.

Förutsägelse

Verklighet \ Förutsägelse	A	B	C	D	E	F	G	H	I	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	
A	25	0	0	0	0	0	0	0	0	0	0	1	2	0	0	0	0	0	1	1	0	1	0	0	80 %
B	0	21	1	0	2	4	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	2	0	0	65 %
C	0	0	25	0	0	2	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	78 %
D	0	1	0	24	0	1	0	1	0	0	1	0	0	0	0	0	1	1	1	0	0	0	1	0	75 %
E	0	2	0	0	22	0	1	0	0	0	0	6	0	0	0	0	0	1	0	0	0	0	0	0	68 %
F	0	1	0	0	1	28	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	90 %
G	0	0	0	0	0	0	22	3	0	0	0	1	0	0	1	0	0	0	1	0	0	0	3	0	70 %
H	1	0	0	0	0	0	5	20	0	0	0	0	1	1	1	0	0	0	2	0	0	0	1	0	62 %
I	0	0	0	0	0	1	0	0	26	0	0	0	0	0	0	1	0	0	0	0	0	0	0	2	86 %
K	0	0	0	0	0	0	0	0	0	24	0	1	0	2	0	0	1	0	0	0	3	1	0	0	75 %
L	0	0	0	1	0	0	1	1	0	0	29	0	0	0	0	0	0	0	0	0	0	0	0	0	90 %
M	1	0	0	0	0	1	0	0	0	0	0	23	1	0	1	0	0	2	0	0	0	0	0	0	79 %
N	0	0	0	0	0	0	1	0	0	0	0	2	26	0	0	0	0	1	1	0	0	0	0	0	83 %
O	1	0	1	0	0	0	0	0	0	0	0	1	1	26	0	0	0	0	0	0	0	2	0	0	81 %
P	0	0	0	1	0	0	1	3	0	0	1	0	0	0	26	0	0	0	0	0	0	0	1	0	78 %
Q	0	0	0	0	0	0	0	2	0	0	1	0	0	0	3	24	0	1	0	0	0	0	1	0	75 %
R	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	23	0	0	2	0	1	3	0	71 %
S	2	0	0	0	6	0	1	0	0	0	0	0	2	0	1	0	0	19	0	0	0	1	0	0	59 %
T	2	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	27	0	0	0	1	0	84 %
U	0	1	0	0	0	1	0	0	0	1	0	0	0	0	0	0	4	0	0	23	2	0	0	0	71 %
V	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	21	7	0	0	67 %
W	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	30	0	0	96 %
X	0	0	2	1	1	1	0	1	0	0	0	1	2	2	0	0	0	0	0	0	0	0	21	0	65 %
Y	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	29	93 %
Felklassificeringsandel	21 %	25 %	13 %	14 %	33 %	28 %	40 %	35 %	10 %	7 %	9 %	37 %	27 %	18 %	21 %	4 %	20 %	24 %	20 %	17 %	19 %	33 %	34 %	6 %	

Känslighet

Figur C.3: Förväxlingsmatris för klassificeringsnätverket *Rust*.

Förutsägelse

Verklighet \ Förutsägelse	A	B	C	D	E	F	G	H	I	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	
A	16	1	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	7	3	0	0	2	0	0	51 %
B	0	25	0	0	0	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	78 %
C	0	0	28	0	0	0	1	0	0	0	1	0	0	2	0	0	0	0	0	0	0	0	0	0	87 %
D	1	0	0	24	0	1	0	0	1	0	1	0	0	0	0	0	1	0	1	0	0	1	1	0	75 %
E	0	1	0	0	25	0	0	0	0	1	0	2	0	0	0	0	0	3	0	0	0	0	0	0	78 %
F	0	1	0	0	0	29	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	93 %
G	0	0	1	0	1	0	23	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5	0	74 %
H	0	0	1	0	1	0	7	19	0	0	0	1	0	1	0	0	0	0	1	0	0	0	1	0	59 %
I	1	0	0	0	0	0	0	0	28	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	93 %
K	0	0	0	1	0	0	0	0	0	22	0	0	1	0	0	0	0	0	0	0	3	5	0	0	68 %
L	0	0	0	0	0	0	0	1	0	0	31	0	0	0	0	0	0	0	0	0	0	0	0	0	96 %
M	0	1	0	0	1	0	0	0	0	0	0	21	5	0	0	0	0	0	0	0	0	0	1	0	72 %
N	0	1	0	0	0	1	0	0	0	0	0	1	27	0	0	0	0	1	0	0	0	0	0	0	87 %
O	0	0	1	1	0	0	0	2	0	0	0	0	0	28	0	0	0	0	0	0	0	0	0	0	87 %
P	1	0	0	0	0	0	1	4	0	0	0	0	0	0	25	2	0	0	0	0	0	0	0	0	75 %
Q	0	0	0	0	0	0	0	1	0	0	1	0	0	0	1	27	0	0	0	0	0	0	2	0	84 %
R	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	26	0	0	3	0	1	0	0	81 %
S	1	0	0	0	6	0	1	0	0	0	0	2	0	0	0	0	0	21	1	0	0	0	0	0	65 %
T	2	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	1	27	0	0	0	0	0	84 %
U	0	0	0	0	0	2	0	0	1	1	0	0	0	0	0	0	6	0	0	21	1	0	0	0	65 %
V	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	26	3	0	0	83 %
W	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	29	0	0	93 %
X	0	0	0	0	0	0	3	1	0	0	0	2	1	1	0	0	0	0	0	0	0	0	24	0	75 %
Y	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	30	96 %
Felklassificeringsandel	27 %	19 %	9 %	7 %	30 %	25 %	36 %	34 %	12 %	12 %	11 %	30 %	22 %	15 %	3 %	6 %	21 %	36 %	18 %	12 %	16 %	32 %	29 %	3 %	

Känslighet

Figur C.4: Förväxlingsmatris för klassificeringsnätverket *Sherlock*.

D

Nätverksstrukturer

Nedan presenteras strukturen för de tre klassificeringsnätverken. Förklaring av kolonnernas rubriker finns på nästkommande sida.

Tabell D.1: Strukturer på klassificeringsnätverk.

Gadget							
lager	*typ*	*filterbredd*	*filterdjup*	*antal filter*	*steglängd*	*data-storlek*	*data-djup*
0	input	–	–	–	–	128	4
1	conv	9	4	64	3	40	64
2	bnorm	1	–	–	1	40	64
3	mpool	3	–	–	2	19	64
4	relu	1	–	–	1	19	64
5	conv	8	64	128	1	12	128
6	bnorm	1	–	–	1	12	128
7	mpool	3	–	–	1	10	128
8	relu	1	–	–	1	10	128
9	conv	6	128	256	1	5	256
10	bnorm	1	–	–	1	5	256
11	mpool	2	–	–	2	2	256
12	relu	1	–	–	1	2	256
13	dropout	1	–	–	1	2	256
14	conv	2	256	512	1	1	512
15	bnorm	1	–	–	1	1	512
16	relu	1	–	–	1	1	512
17	dropout	1	–	–	1	1	512
18	conv	1	512	24	1	1	24
19	bnorm	1	–	–	1	1	24
20	relu	1	–	–	1	1	24
21	softmxl	1	–	–	1	1	1
Rust							
lager	*typ*	*filterbredd*	*filterdjup*	*antal filter*	*steglängd*	*data-storlek*	*data-djup*
0	input	–	–	–	–	64	4
1	conv	9	4	30	1	56	30
2	mpool	4	–	–	4	14	30
3	relu	1	–	–	1	14	30
4	conv	7	30	50	1	8	50
5	mpool	2	–	–	2	4	50
6	relu	1	–	–	1	4	50
7	conv	4	50	60	1	1	60
8	relu	1	–	–	1	1	60
9	dropout	1	–	–	1	1	60
10	conv	1	60	24	1	1	24
11	softmxl	1	–	–	1	1	1
Sherlock							
lager	*typ*	*filterbredd*	*filterdjup*	*antal filter*	*steglängd*	*data-storlek*	*data-djup*
0	input	–	–	–	–	64	4
1	conv	9	4	64	1	56	64
2	bnorm	1	–	–	1	56	64
3	mpool	4	–	–	4	14	64
4	relu	1	–	–	1	14	64
5	conv	7	64	128	1	8	128
6	bnorm	1	–	–	1	8	128
7	mpool	2	–	–	2	4	128
8	relu	1	–	–	1	4	128
9	dropout	1	–	–	1	4	128
10	conv	4	128	256	1	1	256
11	bnorm	1	–	–	1	1	256
12	relu	1	–	–	1	1	256
13	dropout	1	–	–	1	1	256
14	conv	1	256	24	1	1	24
15	bnorm	1	–	–	1	1	24
16	relu	1	–	–	1	1	24
17	softmxl	1	–	–	1	1	1

lager Beskriver ordningen på de lager som används.

typ Beskriver vilken typ av lager det är. Här är **conv** faltningslager, **bnorm** batch normalization-lager, **mpool** pooling-lager, **relu** aktiveringslagret *Rectified linear unit*, **dropout** dropout-lager och **softmxl** ett *soft max loss*-lager.

filterbredd Anger filtrets bredd.

filterdjup Anger filters djup. I våra fall så är första lagrets filterdjup 4, då indatan är en bild i färgrymden LUV tillsammans med en binär segmenteringsmask.

antal filter Beskriver hur datan förgrenar sig vid faltning.

steglängd Anger steglängden för lagrets operation.

datastorlek Anger bredd och höjd för data. Att datastorleken till exempel är 64 för *input* innebär att nätverket förväntar sig att en bild som indata är 64 pixlar bred och hög.

datadjup Anger djupet på data efter att lagret utfört sina beräkningar.

E

Evaluering av hela bilder med ett pixel-för-pixel-nätverk

Att formulera ett nätverk som reducerar en mängd indata till en uppsättning *features* är relativt enkelt. I detta arbete användes exempelvis ett nätverk som reducerar en pixel med en 29×29 pixlars omgivning till 100 *features* som sedan reducerades till 2 klasser.

Evaluering med ett sådant nätverk kan göras naivt genom att evaluera alla pixelområden för sig. Detta tillvägagångssätt kan dock vara långsamt och resulterar oundvikligen i att upprepade beräkningar.

Ett tänkbart alternativ skulle vara att skapa ett nätverk som tar indata med storlek $n \times n$ och producerar $n \times n$ uppsättningar av särdrag. Men ett sådant nätverk skulle behöva tränas om från grunden, vilket ofta kan vara tidskrävande.

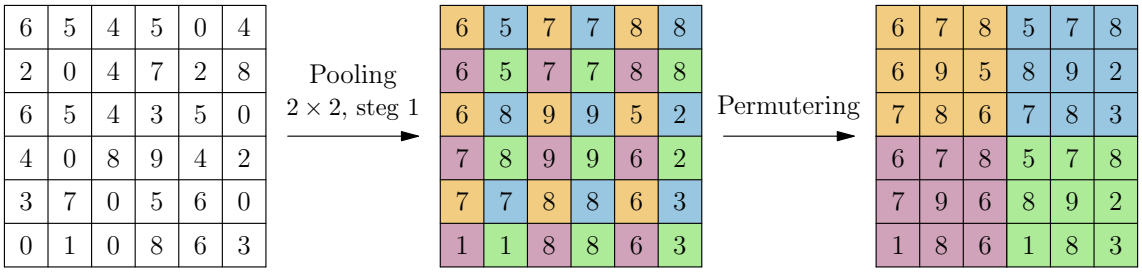
En annan idé är att anpassa det existerande nätverket för att kunna arbeta på hela bilder. Det är denna idé som kommer presenteras i kommande sektioner.

E.1 Problemet med nätverket

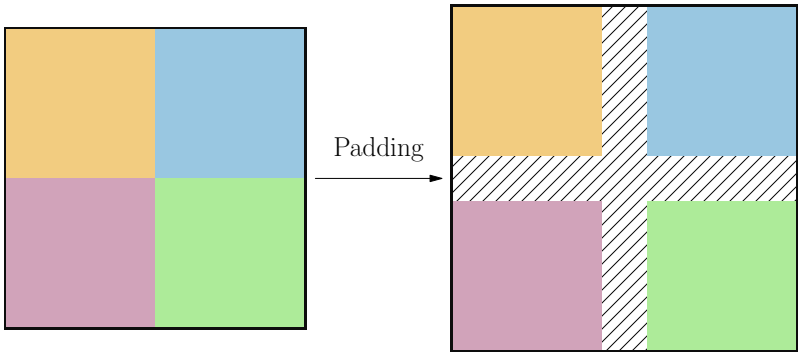
Vad är problemet med att direkt evaluera en hel bild med nätverket? Faltningslagren kan enkelt producera resultat med samma storlek som indatan givet att padding görs korrekt och de fullt kopplade lagren blir till faltningslager som fungerar precis om väntat. Det som vållar problem är pooling-lager.

Ta som exempel segmenteringsnätverket som beskrivs i sektion 3.1.4.1. Problemet med pooling kan enkelt ses genom att studera evaluering av de första 2×2 pixlarna. Pixlarna kommer ha olika förskjutning på det rutnät för vilket poolingen görs, så när en hel bild skickas in i nätverket används enbart en av fyra poolingar vid varje lager. En stor mängd data förkastas.

En metod för att lösa problemet är att skapa grenar vid varje pooling-lager, där varje gren motsvarar en av de möjliga (typiskt fyra) poolingkonfigurationerna, och propagera varje gren för sig. Detta skulle fungera, men involverar många nätverksevalueringar. Processen visar sig kunna göras på sådant sätt att varje lager i nätet används exakt en gång.



Figur E.1: Poolingprocess för evaluering av hel bild.



Figur E.2: Padding (streckat) läggs mellan delbilder för att undvika att de påverkar varandra vid faltning.

E.2 En praktisk lösning

Det tillvägagångssätt som implementerades för segmenteringen börjar med att stycka upp segmenteringsnätet efter varje poolinglager så att tre nätverk N_1, N_2, N_3 fås. Parametrar för nätverken ändras så att alla faltningsnät paddar med nollor så att storlek bevaras och så att pooling sker med steglängd ett så att storlek bevaras.

Efter evaluering av N_1 fås en fullstor bild med alla möjliga poolingresultat. Påföljande faltningar ska dock göras på mindre bilder poolade på vanligt sätt med steglängd två. Detta kan åstadkommas genom att inse att varje block av 2×2 pixlar innehåller de fyra olika poolingmöjligheterna, så genom att flytta runt pixlar kan de fyra olika poolade bilderna skapas, schematiskt exempel kan ses i Figur E.1. Den datamatris som nu fås innehåller nu all data som ska existera efter det första poolinglagret.

Det är värt att nämna att det inte är en bra idé att direkt skicka in denna matris till N_2 direkt då faltningar får delbilderna att kombineras (något som inte skall få ske). Detta löses enkelt genom att lägga in padding med nollor mellan delbilderna som i Figur E.2.

När väl detta är gjort kan exakt samma process göras för att evaluera N_2 och därefter kan N_3 evalueras. Den data som då fås ut består av 16 olika delbilder för de 16 olika poolingkombinationerna som uppstår för två 2×2 poolingar. Det kvarstår nu att korrekt kombinera dessa bilder tillbaks till den fullstora segmenteringen, exakt utformning av en sådan metod lämnas som övning till läsaren.

F

Källkod

Kod som används för projektet finns tillgänglig på ett git repository som hittas på `http://bitbucket.org/neuralkandidat/tstpublic`. Koden är inte nödvändigtvis renskriven eller välkommenterad och presenteras ”as is”.