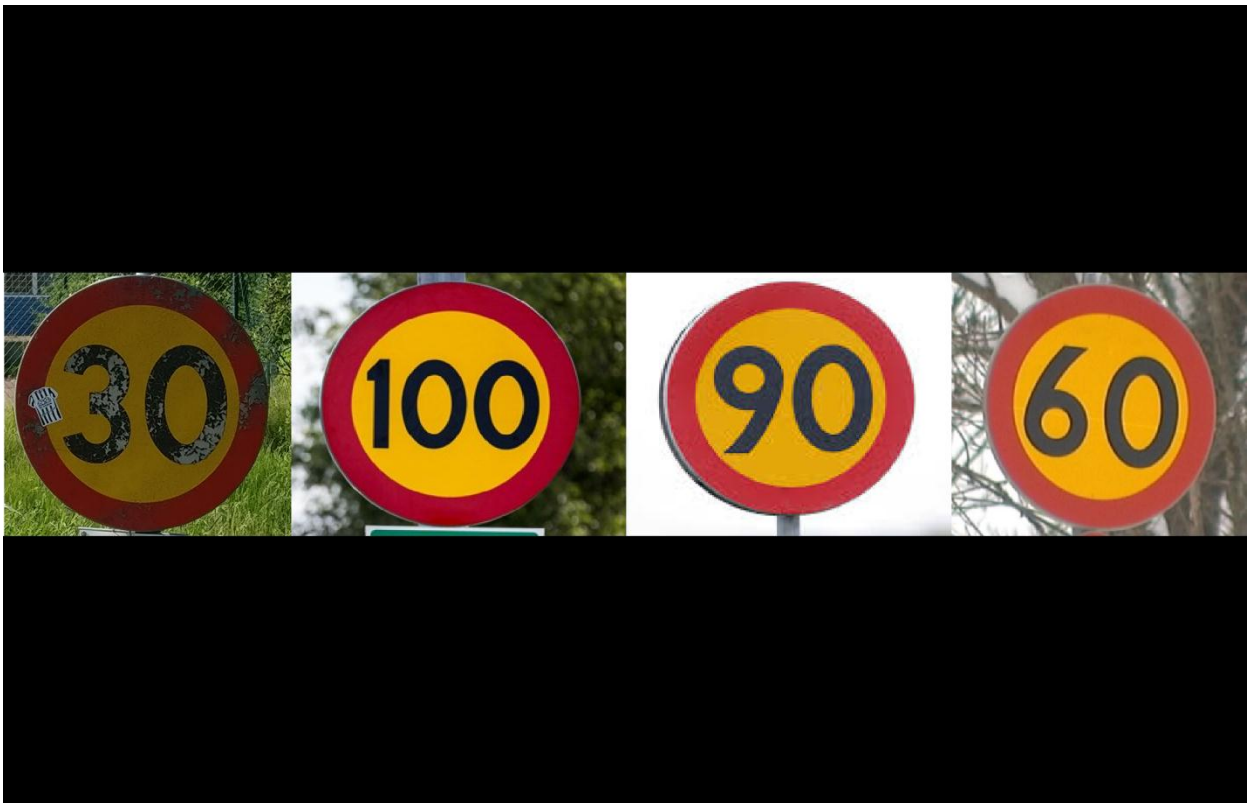
CHALMERS



Vägskyltsdetektion i OpenCV med Haar och LBP

Examensarbete inom högskoleingenjörsprogrammet Elektroingenjör

PONTUS KARLSSON

Institutionen för Elektroteknik
CHALMERS TEKNISKA HÖGSKOLA
Göteborg, Sverige 2018

Förord

Denna rapport handlar om det examensarbete på Chalmers tekniska högskola som jag har ägnat vårterminen 2018 åt att utföra. Arbetet utfördes med hjälp av Broccoli AB. Jag vill tacka Björn Bergholm och Petra Panzar för att ha givit mig möjligheten att utföra detta arbete och för det stöd de har givit mig på vägen.

Sammanfattning

Bildklassificering och bildigenkänning har länge funnits. Nu när bilindustrin utvecklat och implementerat dessa system i vissa av sina nya modeller så lämnas äldre modeller utanför utvecklingen. Slutmålet är att dessa system skall utvecklas till delsystem i självkörande bilar. Om ett liknande system kan utvecklas för att implementeras på äldre bilmodeller så kan dessa bilars livstid förlängas och återförsäljningsvärde ökas. I denna rapport redogörs för möjligheten att använda open-source material för att implementera ett jämförbart system. Resultatet redogör förträffsäkerheten för de två algoritmer som används, HAAR och LBP. Goda resultat har uppnåtts, där systemen har en detektionsnivå mellan 86% och 100% enligt egna tester.

Abstract

Image classification and object detection has existed for quite some time. But now that the car industry is developing these kinds of systems for use in their new vehicles the older vehicles are quickly becoming outdated. The purpose of these systems is to implement a completely autonomous driving car. If a comparable system could be developed and implemented in older car models the lifetime and resale value could rise significantly. In this report the possibility of using open-source software for this purpose is investigated. The result compares the accuracy of the two OpenCV compatible algorithms, Haar and LBP. Results have been good with the systems showing a success rate between 86% and 100%, according to tests conducted.

Terminologi

Då Computer Vision industrin främst uttrycker sig i engelska termer har valet gjorts att denna rapport gör detsamma. Detta betyder att vissa termer är på engelska. Här är en lista med förklaringar på vissa tekniska termer som används i rapporten.

Feature
En karaktäristik som beskriver ett område.

Feature Vector
En vektor som innehåller samtliga features för en bild.

Träning
Träning innefattar utvecklingen av ett Machine-Learning system genom en stor mängd data.

Histogram
Ett histogram är en grafisk representation av koncentrationen av numerisk data.

Positiva och Negativa bilder
Positiva bilder innefattar de bilder som innehåller objektet systemet skall tränas för att hitta. Negativa bilder innefattar de bilder som inte innehåller objektet som systemet skall tränas för att hitta.

False Positives
Fall där systemet gör ett felaktigt påstående om närvaron på det objekt det tränats för att hitta.

LBP
LBP står för Local Binary Pattern och är en av de algoritmer som undersöks i denna rapport.

Haar-Kaskad
Haar-Kaskad, eller Haar, är den andra algoritmen som undersöks i denna rapport. Den är döpt efter Alfréd Haar, som 1909 upptäckte Haar-Wavelets.

Innehåll

1. Inledning

1.1 Bakgrund

Maskininlärning och bildklassificering är forskningsområden där det skett en stor ökning av intresse. Förståelsen för dess potential ökar dagligen och det finns ett brett användningsområde. En av industrierna som blivit väldigt intresserade av detta är fordonsindustrin. Målet är att kunna ersätta mänskliga förare med automatiska system. En del av dessa system är trafikskyltsigenkänning, en vital del för en förares förmåga att ta beslut i trafiken.

Idag finns det system för trafikskyltsigenkänning i nya bilmodeller från en stor majoritet av biltillverkare. Dessa system är främst integrerade i nya bilmodeller. Detta kommer skapa en stor klyfta mellan moderna och åldrade bilar. Om ett modulärt system som kan implementera dessa funktioner i samtliga fordon kan utvecklas så kan potentiellt ett billigare alternativ tas fram för konsumenter med mindre köpkraft.

1.2 Syfte

Syftet med denna rapport är att framställa ett system för identifiering av trafikskyltar och med hjälp av open-source programmet OpenCV designa neurala nätverk för att lösa detta problem. Detta system skall implementeras på en Raspberry Pi för att utvärdera prestandan.

1.3 Problemformulering

Rapporten hanterar svårigheterna att framställa ett tillräckligt system för identifiering av trafikskyltar. Systemet har som uppgift att identifiera trafikskyltar i bilder och videor.

Igenkänningen delas upp i två delproblem. Införskaffandet av referensbilder och val av träningsinställningar.

Systemet skall uppfylla följande specifikationer:
Minst 80% igenkänningsfaktor
Max 20% false positives
30 bildanalyser per sekund

De träningsinställningar som skall testas är följande
Upplösning
Algoritmerna LBP och Haar
Träningsprescision

Systemet skall främst fungera för skyltar på korta avstånd, under 20m, för att fokusera på skyltar fordonet nyligen passerat, inte de den kommer passera.

1.4 Avgränsningar

Under arbetets gång kommer systemet endast designas för att identifiera svenska hastighetsskyltar. Denna begränsning sker då det är tidskrävande att införskaffa tillräckligt med referensbilder till en stor mängd olika typer av skyltar. Systemet kommer utvecklas för hantera skyltar på avstånd upp till 20m, men detektioner på längre avstånd kommer inte förkastas. För att ytterliggare begränsa bredden på problemet kommer textdetektion, alltså förmågan att förstå vilken hastighetsskylt som hittats, ske genom programmet TesseractOCR.

2. Teknisk Bakgrund

2.1 Histogram of Oriented Gradients

Histogram of Oriented Gradients (HOG) är en feature descriptor som används i vissa computer vision resurser, exempelvis OpenCV och MATLAB. En feature descriptor är en förenklad version av en bild där överflödig information är bortfiltrerat. En HOG vektor konverterar en bild array med typiska storleken Bredd * Höjd * 3 (Färgkanaler) till en vektor med längden n. [1] En HOG vektor kan beräknas för andra typer av bildrepresentationer men dessa hanteras ej i denna rapport. HOG vektorn visar fördelningen av riktade gradienter som features. Gradienter och dess riktningar fungerar som en kantdetektion då gradientens magnitud är större vid kanten av ett objekt. En HOG vektor beräknas genom att först filtrera bilden vertikalt och horizontellt för att normalisera ljusintensiteten i bilden. Magnituden och riktningen på gradienterna beräknas med följande formler.

$$g = \sqrt{{g_x}^2 + {g_y}^2}$$

$$\Theta = arctan(\frac{g_y}{g_x})$$

Där g är magnituden och Θ är riktningen på gradienten.

Figur 2.1.1: Representation av gradientens magnitud (g)

Dessa magnituder kombineras sedan till ett histogram. Här delas bilden upp i celler och gradientriktningar beräknas för cellerna. Gradientriktningarna representeras endast i 0-180° då det tillåter spegelvändning av objektet utan att skada igenkänningsfaktorn.

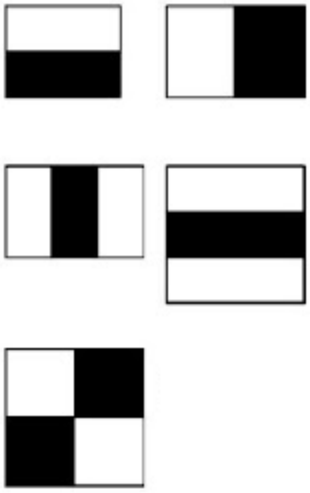
Efter detta är det vanligt att normalisera block av flera celler, då varje individuell cell kan påverkas av ljusnivån. Normaliseringen sker genom att dela intensiteten på varje pixel med två, alltså göra bilden mörkare. Detta halverar även effekten som ljusnivån har mellan två olika gradienter med två.

Efter detta kan blocken beräknas till en HOG feature vektor genom att se på de histogram som beräknats. Att visuellt representera detta är svårt, men i figur 2.1.2 kan man observera att de röda linjerna följer formen av personen någorlunda väl, men med något låg upplösning.

Figur 2.1.2: En visuell representation av en HOG feature vektor

2.2 Haar-Cascade

Haar-Cascade är en klassifieringsalgoritm som används för att detektera objekt i bilder. Algoritmen är fokuserad runt så kallade Haar-Features, vilket är en metod för att förenkla bilder till vissa karakteristiker. Dessa karaktarestiker framställs genom att ta genomsnittliga pixelvärdet (i gråskala) i ett område, och genom skillnader i genomsnittsvärden mellan områden framställa en feature. Det finns flera olika typer av Haar-features, exempelvis kanter, linjer, schackbräde och punkter. Det finns även roterade versioner av dessa features. Alla dessa features är fördefinierade men det finns ingen teoretisk begränsning på hur många och vilka features som skall användas. I denna rapport används BASIC samlingen, vilket ger en fråga över hur bra systemet kommer vara på att känna igen mer komplexa objekt. Basic samlingen innehåller följande mönster:



Rad ett är kanter. Där pixelsamlingen går från ljus till mörk, ellet tvärtom.
Rad två är linjer, där pixelsamlingen går från ljus till mörk tillbaka till ljus, eller tvärtom.
Rad tre är ett schackbrädemönster. Detta förväntas inte komma till stor användning i denna rapports implementation.

Målet med Haar-features är att göra detektionen mycket mindre beräkningsintensiv genom förenkla ett objekt till några features, istället för att individuellt analysera varje pixel.

Figur 2.2. Visualisering av HAAR-features

2.3 Local Binary Patterns

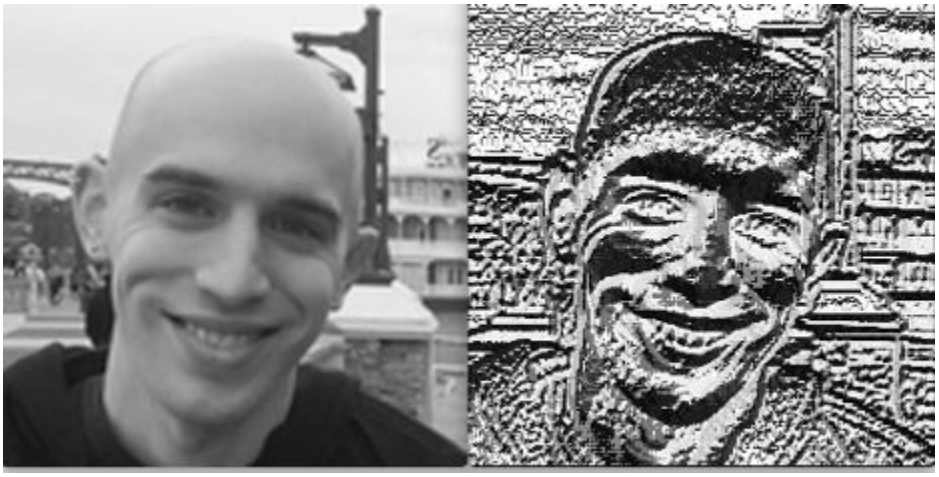
Local Binary Patterns, liknande Haar-Cascade, är en klassifieringsalgoritm.
En Local Binary Patterns (LBP) feature vektor, i sin simplaste form, beräknas i följande steg.

Först delas en bild in i flera celler (exempelvis 16x16 pixlar). Varje pixel i cellen jämförs med sina 8 närliggande grannar. Där pixelns storlek är större än den jämförde pixelns värde ges binära värdet 0, annars 1. Detta ger ett 8-bitars binärt nummer för varje pixel.

Ett histogram beräknas för varje cell för frekvensen av varje binära nummer som uppstår. Detta histogrammet kan agera som en feature vektor. Efter detta kan en normalisering, och slutligen en sammankoppling av samtliga celler ske. Detta ger en feature vektor för hela bilden.

LBP har visats vara en kraftfull algoritm för att känna igen texturer och har visat mycket goda resultat när den använts i samarbete med ett HOG system. [2]

I OpenCV 3.0 skapades stöd för att använda LBP istället för Haar, om så önskas.



Figur 2.3. Visualisering av LBP.

2.4 OpenCV

2.4.1 Createsamples

OpenCV_Createsamples programmet tillkommer i OpenCVs installation. Detta systemet används för att förbereda positiv data inför träning i OpenCV.

Utöver att förbereda existerande data inför träning kan detta system även artificiellt framställa mer data inför träning. Detta då det är mycket tidskrävande att framställa en sådan stor mängd bilder manuellt. Dessa artificiellt framställda bilder kan variera i två olika aspekter, distordering och rotering. Distordering framställs genom att ta bilder från den negativa datainsamlingen och från den beräkna en distorsion som placeras på den nya positiva bilden. Rotering sker på alla de tredimensionella axlarna.[3]

Figur 2.4.1: Exempel på hur en bild kan distorderas av OpenCV_createsamples för att emulera mer träningsdata.

Det är viktigt för en användare att inte skapa positiva bilder som är för surrealistiska för att kunna vara användbara i träningsprocessen. För säkerställa sig om detta så finns det diverse inställningar som användaren kan nyttja.

-maxXangle, -maxYangle, -maxZangle är inställningar som styr hur mycket programmet får rotera på existerande bilder i framställandet av nya bilder. Värden anges i radianer.

-w, -h definierar vilken höjd och bredd som den skapade träningsdatan skall vara. Om de positiva inputbilderna till createsamples är i en annorlunda bredd:höjd ratio kommer bilderna stretchas för att passa in i inställningarna.

-bg hänvisar till bakgrundsbilden eller bilderna som används för slumpmässig distordering av de positiva inputbilderna.

-img hänvisar till source bilden. Programmet kan endast distordera en positiv bild i taget. För att kringgå denna begränsning har ett program i språket Perl använts för att genomföra korrekta kommandon till Createsamples programmet, samt att sedan kombinera de positiva vektorfiler som skapats.

-maxidev, eller max_intensity_deviation är maximala intensitetsskillnaden mellan pixlar i positiva inputbilder

2.4.2 TrainCascade

OpenCV_Traincascade programmet används för att träna neurala nätverk för algoritmerna Haar och LBP (se 2.2 och 2.3) för igenkänning. OpenCV_Traincascade använder sig av Adaboost systemet för att genomföra träningsprocessen snabbare.

Adaboost, eller Adaptive-Boost, algoritmen är ett försök att snabba upp den annars långsamma träningsprocessen för att träna neurala nätverk av denna typ. Detta sker genom att träna flera nätverk som sedan kombineras till det slutgiltiga nätverket. Adaboost ger olika vikt till de mindre nätverken i denna kombinationen beroende på hur framgångsrikt de varit på att detektera objektet utan att ge fel. Vikten som ges till varje mindre klassificerare definieras med följande ekvation. [4]

$$\alpha_t = \frac{1}{2} * ln(\frac{1 - \epsilon_t}{\epsilon_t})$$

Där ϵ representerar Error rate, hur ofta klassificeraren har fel, och α representerar vikten som ges till klassificeraren.

En intressant karakteristik i denna ekvation är att en klassificerare med en hög error rate, alltså en klassificerare som har fel mer än den har rätt, fortfarande används. Den ges en negativ vikt i det kombinerade nätverket, vilket betyder att det slutgiltiga nätverket kommer välja att gå emot vad den mindre klassificeraren ger som resultat. Enda gången som en klassificerare helt förkastas är om den ger exakt 50% error rate, vilket betyder att resultatet är en total gissning. En annan intressant karakteristik är att vikten ökar exponentiellt ju närmre error rate är till 0, vilket betyder att en väldigt lyckad klassificerare kan dominera det kombinerade nätverket.

Traincascade innehåller många inställningar för att framställa olika resultat. De tre parametrar som är i fokus i denna rapport är följande.

-AcceptanceRatioBreakValue används för att bestämma hur precist tränat modellen skall vara. När systemet når det bestämda värdet så avslutas träningsprocessen. Ett för precist tränat system kommer ha svårigheter att känna igen någonting, och ett för oprecist tränat system kommer ge många false positives.
-sampleWidth och -sampleHeight, representerar upplösningen på positiva samples och påverkar hur många features som analyseras. Vilken slutgiltig påverkan detta har varierar per algoritm, men med mer data att analysera kommer träningstiden öka skarpt med upplösningen. Dessa parametrar måste vara samma som upplösningen på den positiva datasamlingen som framställts från OpenCV_Createsamples (se 2.4.1).
-FeatureType bestämmer vilken typ av features systemet kommer tränas efter, Haar eller LBP. Båda dessa typer kommer testas i denna rapport.

Andra inställningar som kan ge stora förändringar på resultat är följande:

-maxFalseAlarmRate vilket definierar den maximala tillåtna false alarm nivån för varje steg i klassificeraren. Alltså hur ofta den får ge felutslag.
-minHitRate, vilket definerar den minsta önskade hit rate nivån för varje steg i klassificeraren. Alltså hur ofta den har rätt på träningsdata som innehåller våra positiva bilder. Totala Hit Rate nivån kan uppskattas till minHitRate^N där N representerar antal steg.
-numStages, maximala antalet cascade steg som skall användas. Inställningen kan användas istället för -AcceptanceRatioBreakValue, då den också representerar hur precist modellen är tränad.
-mode, väljer vilken samling av Haar-features som analyseras i fallet att man använder Haar algoritmen. I denna rapport används endast BASIC samlingen av features.

```
===== TRAINING 11-stage =====
<BEGIN
POS count : consumed   850 : 948
libpng warning: iCCP: known incorrect sRGB profile
NEG count : acceptanceRatio    1800 : 0.00210681
Precalculation time: 25.996
+----+---------+---------+
|  N |    HR   |    FA   |
+----+---------+---------+
|   1|        1|        1|
+----+---------+---------+
|   2|        1|        1|
+----+---------+---------+
|   3|        1|        1|
+----+---------+---------+
|   4|        1|        1|
+----+---------+---------+
|   5|        1|        1|
+----+---------+---------+
|   6|        1|        1|
+----+---------+---------+
|   7|        1|        1|
+----+---------+---------+
|   8| 0.998824| 0.946667|
+----+---------+---------+
|   9| 0.998824| 0.952222|
+----+---------+---------+
|  10| 0.995294|    0.835|
+----+---------+---------+
```

Fig 2.4.2. Exempel på hur träningsprocessen ser ut. HR står för Hit Rate, FA står för False Alarm.

2.5 TesseractOCR

TesseractOCR är en open-source Optical Character Recognition (OCR) motor vars utveckling påbörjades 1984 hos Hewlett-Packard. 2005 gjorde Hewlett-Packard motorn open-source och mycket utveckling har sedan skett. Tesseract är en motor för att känna igen bokstäver och nummer i en bild och har många algoritmer för att genomföra detta syftet. Metoder för att bryta upp bokstäver som sitter ihop, är deformerade eller på annat sätt distorderade är något Tesseract varit framgångsrik med att genomföra. [5]

3. Metod och Material

3.1 Datahantering

Neurala nätverk kräver mycket data för att framställa en tillräckligt bra modell för detektion. I detta kontextet är data bilder på hastighetsskyltar. Då framställandet av många individuella bilder är mycket tidskrävande togs beslutet att använda algoritmer för att artificiellt skapa fler positiva bilder.

52 stycken bilder på olika hastighetsskyltar i varierande kvalité och ljusförhållanden samlades in genom internet och kamerafotografering. Med hjälp av OpenCV CreateSamples (se 2.4.1) algoritmer ökades denna mängd till 1100. Anledningen till att artificiell framställning av träningsdata användes är att det är mycket tidskrävande att ta fram en god mängd bilder, balanserade mellan olika hastighetsskyltar, i olika ljusförhållanden och i olika nivåer av slitage. Detta kan dock orsaka vissa problem. I fallet att det är för många och för overkliga artificiella bilder så kan systemets förmåga att känna igen verkliga hastighetsskyltar påverkas.

Även negativa bilder behövs för att träna systemet. Här används det redan existerande biblioteket från ImageNet [6] med en stor mängd bilder. Vid träning användes 1800 negativa bilder. Dessa bilder inkluderar ett stort antal olika objekt.

Figur 3.1: Exempel på de negativa bilder som användes vid träning. [6]

3.2 Textigenkänning

För att systemet skall lyckas analysera vilken hastighetsskylt som hittats så används TesseractOCR. Det område som hittats av systemen från 3.3 skickas in i TesseractOCRs algoritm för analys.

3.3 Raspberry Pi

Det fullständiga systemet implementerades på en Raspberry Pi 3 model B med Raspian operativsystem och samtliga relevanta paket. Systemet innehåller även en Raspberry Pi Camera v2 modul för att framställa ett bildinput till systemet. Då prestandatillgängligheten är låg på en så billig hårdvarulösning körs kameran med 640x480 pixlar i upplösning med en bildfrekvens på 30 bilder per sekund.

Figur 3.3: Raspberry Pi 3 model B med kameramodul.

3.3 Test

För att säkerställa att träningsmetoderna är goda genomfördes en mängd tester. Detta krävde flera tränade system med varierande inställningar. Nedan är samtliga variationer av system som tränades och testades. För förklaring bakom dessa inställningar se kapitel 2.4.2

Tabell 3.3: Samtliga testsystem

Upplösning positiva samples	AcceptanceRatioBreakValue	Algoritm
20x20	$10e^{-5}$	Haar
20x20	$5e^{-5}$	Haar
20x20	$1e^{-5}$	Haar
20x20	$10e^{-5}$	LBP
20x20	$5e^{-5}$	LBP
20x20	$1e^{-5}$	LBP
25x25	$10e^{-5}$	Haar
25x25	$5e^{-5}$	Haar
25x25	$1e^{-5}$	Haar
25x25	$10e^{-5}$	LBP
25x25	$5e^{-5}$	LBP
25x25	$1e^{-5}$	LBP
30x30	$10e^{-5}$	Haar
30x30	$5e^{-5}$	Haar
30x30	$1e^{-5}$	Haar
30x30	$10e^{-5}$	LBP
30x30	$5e^{-5}$	LBP
30x30	$1e^{-5}$	LBP

Testet involverar 25st stillbilder där det finns totalt 36st hastighetsskyltar, målet är att systemet skall detektera så många som möjligt utan att presentera false positives. Samtliga av dessa bilder är på nära avstånd, under 10m, med goda ljusförhållanden.

Figur 3.3.1: Några av de bilder som användes i detektionstestet.

Ytterligare ett test kommer genomföras. Detta för att undersöka systemens förmåga att känna igen skyltar på långt avstånd. I detta test så analyserar varje system 18st olika bilder, alla på samma skylt vid progressivt längre avstånd, ca 2m till 20m. Testet avslutas då systemet inte längre känner igen en skylten.

Figur 3.3.2: En av de bilder som analyseras i avståndstestet.

4. Resultat

4.1 Test

Testen gav fölande resultat

Tabell 4.2.1: Resultat av detektionstest

Upplösning positiva samples	AcceptanceRatio BreakValue	Algoritm	Detekterade Skyltar	% Detekterade Skyltar	Antal False Positives	% False Positives	Range
20x20	1.00E-04	Haar	35	97.22	12	25.53	20+
20x20	5.00E-05	Haar	34	94.44	6	15	20+
20x20	2.00E-05	Haar	34	94.44	4	10.53	20+
20x20	1.00E-04	LBP	32	88.89	1	3.03	20+
20x20	5.00E-05	LBP	32	88.89	1	3.03	20+
20x20	2.00E-05	LBP	31	86.11	0	0	20+
25x25	1.00E-04	Haar	36	100	37	50.68	20+
25x25	5.00E-05	Haar	36	100	12	25	20+
25x25	2.00E-05	Haar	36	100	4	10	20+
25x25	1.00E-04	LBP	33	91.67	4	10.81	20+
25x25	5.00E-05	LBP	33	91.67	0	0	20+
25x25	2.00E-05	LBP	32	88.89	0	0	20+
30x30	1.00E-04	Haar	36	100	40	52.63	20+
30x30	5.00E-05	Haar	36	100	17	32.08	20+
30x30	2.00E-05	Haar	35	97.22	5	12.5	20+
30x30	1.00E-04	LBP	36	100	5	12.2	20+
30x30	5.00E-05	LBP	35	97.22	0	0	20+
30x30	2.00E-05	LBP	34	94.44	0	0	20+

4.2 Hårdvaruimplementation

Hårdvaruimplementationen av systemet på en Raspberry Pi 3 model B påvisar det relativt höga prestandakravet för att genomföra de beräkningar som krävs. Trots den låga upplösningen har systemet svårighet att analysera samtliga av de 30 bilder per sekund som produceras av kameran. Dessa prestandaproblem varierade mellan system. Där systemen med lägre upplöst träningsdata samt de som använde LBP algoritmen påvisade en högre bildfrekvens än de andra system.

Minnesbegränsningar är även ett problem. OpenCV, Tesseract och alla de relevanta paket som behöver installeras för att det Linux-baserade operativsystemet skall kunna hantera programmet resulterar i att det medföljande 8GB minneskortet till Raspberry Pi 3 model B är otillräckligt.

5. Diskussion och Slutsats

5.1 Raspberry Pi och prestandakrav

Lösningen på problemet som undersöks i denna rapport är något beräkningstung. Detta leder till att en Raspberry Pi är undermålig från rapportens prestandaperspektiv. Vad som krävs i form av bildfrekvens varierar med fordonets hastighet och hur bra systemet är på att detektera skyltar. Att systemet inte kunde analysera 30 bilder per sekund är inte nödvändigtvis ett problem, och är det ett problem så bör det bara uppstå vid högre hastigheter. Med ett system som reglerar inputbildens upplösning så kan man potentiellt kringgå detta problem.

Om systemet skall hålla en hög frekvens av analyserade bilder utan att minimera bildens upplösning bör vissa optimeringar genomföras. På grund av tidsbegränsningar har inte några optimeringsalternativ undersökts i rapporten men användningen av ett filter som filtrerar ut färger som inte är möjliga för en hastighetsskylt att utstråla hade potentiellt kunnat underlätta beräkningskraven drastiskt. I en praktisk implementation på ett fordon så kommer en del av den analyserade bilden sällan innehålla en hastighetsskylt som är nära nog att kännas igen av systemet. Om man väljer att inte analysera denna del av bilden kan man enkelt lätta på prestandakraven.

En alternativ hårdvarulösning hade kunnat nyttja Nvidias Jetson moduler. Stora fördelen med detta är att man kunnat använda CUDA acceleration på den beräkningstunga analysen som genomförs av systemet. Detta bör i teorin ge mycket högre prestanda.

Min slutgiltiga åsikt är att en Raspberry Pi går att använda till detta ändamål, men mycket hänsyn bör då ges till optimeringen av programmet som skall köras.

5.2 Testresultat

Ett uppenbart övertag som LBP har över Haar är tiden som krävs för träningsprocessen. När modeller tränades till testerna så var det uppenbart att modellerna som använde LBP tränades enormt mycket snabbare. I träningsprocessen för modellerna med upplösningen 30x30 pixlar och precisionen 10e^-5 var tidsminskningen över 80% jämfört med Haar modellens träningstid.

I detektionstestet (se Fig 4.2.1) kan man observera mycket färre false positives i LBP systemen. Där systemen visade mellan 0% och 12.2% false positives. Detta är ett starkt övertag över Haar systemen som påvisade 10.53% till 50.68% false positives. Då specifikationerna kräver en maximal false positive nivå på 20% klassas flera av Haar systemen som undermåliga.

Men Haar hade ett övertag i antalet detekterade skyltar, där de Haar-baserade systemen varierade mellan 94.44% och 100% detekterade skyltar i testet, jämfört med LBP systemens 86.11% till 100%. I en praktisk användning så varierar det hur skadligt det är att missa en skylt vid analys av en bild. Om fordonet rör sig i mycket hög hastighet finns det en risk att systemet bara har tid att analysera en eller två bilder innan fordonet passerat skylten. Samtliga system uppfyller 80% kravet enligt specifikationen.

En högre upplösning på positiva träningsdatan orsakade att systemen blev mindre precisa, vilket resulterade både i fler detekterade skyltar och fler false positives. Detta verkar påverka Haar systemen mer än LBP systemen, men ytterligare tester krävs för att säkerställa detta.

Högre träningsprescision orsakade, oförvånande, att systemet blev mer precist. Intressant var att den stora skillnaden i precision inte påverkade antalet detekterade skyltar särskilt starkt. Detta kan förklaras av det goda bildförhållandena som detektionstestet innehöll. I tester med sämre bildförhållanden förväntas denna inställning ha större negativ påverkan.

Avståndstestet gav tyvärr inte några användbara resultat. Samtliga system kände igen skylten i fråga på alla avstånd utan några felutslag. Mer tester krävs för att säkerställa om systemen är framgångsrika på avstånd eller om detta test bara var undermåligt.

5.3 Hållbarhet och etik

Systemet som designats implementerats på en prototyp är inte utan miljöeffekter. Hårdvarulösningen som använts i denna rapport måste produceras, och om systemet skall implementeras på stor skala betyder detta storskalig produktion av beräkningshårdvara. Denna hårvara kan innehålla olika metaller så som koppar, guld, palladium, platinum och silver. [7] Detta kan marignellt höja den globala resurskonsumptionen av dessa metaller. Återvinningsgraden på sådan hårdvara är även låg, vilket kan orsaka ett långtsiktigt hållbarhetsproblem.

Etiskt så finns risken att systemet implementeras i ett autonomt fordon, vilket medför risken för skada på både egendomar och personer. Förhoppningsvis så tas detta beslut inte utan fortsatt utveckling av systemet så att man kan garantera minimala faror för personer och objekt både i, och utanför, detta teoretiska autonoma fordon.

Källhänvisning

[1] Learn OpenCV. [Online] Tillgänglig på:
https://www.learnopencv.com/histogram-of-oriented-gradients/
Hämtad 25-Maj-2018

[2] An HOG-LBP human detector with partial occlusion handling, Xiaoyu Wang, Tony X. Han, Shuicheng Yan, 2009.

[3] OpenCV User Guide [Online] Tillgänglig på:
https://docs.opencv.org/2.4.13.2/doc/user_guide/ug_traincascade.html
Hämtad 26-Maj-2018

[4] Explaining Adaboost, Robert E. Schapire [Online] Tillgänglig på:
http://rob.schapire.net/papers/explaining-adaboost.pdf
Hämtad 6-Juni-2018

[5] An Overview of the Tesseract OCR Engine, Ray Smith, Google Inc. [Online] Tillgänglig på:
https://static.googleusercontent.com/media/research.google.com/sv//pubs/archive/33418.pdf
Hämtad 28-Maj-2018

[6] ImageNet, image database [Online] Tillgänglig på:
http://image-net.org/index
Hämtad 30-Mars-2018

[7] “Obsolete Computers, ’Gold Mine,’ or High-Tech Trash? Resource Recovery from Recycling”, United States Geological Survey, [Online] Tillgänglig på:
https://pubs.usgs.gov/fs/fs060-01/fs060-01.pdf
Hämtad: 19-Juni-2018

Bildkällor

Figur 2.1.1 och 2.1.2:
Learn OpenCV, hämtad 05-juni-2018 [Online]
https://www.learnopencv.com/histogram-of-oriented-gradients/

Figur 2.2:
OpenCV documentation, hämtad 06-juni-2018 [Online]
https://docs.opencv.org/3.4.1/d7/d8b/tutorial_py_face_detection.html

Figur 2.3:
Pyimagesearch, hämtad 06-Juni-2018 [Online]
https://www.pyimagesearch.com/2015/12/07/local-binary-patterns-with-python-opencv/