



CHALMERS

Autonom körning i realtid

Jämförelse av olika artificiella neurala nätverk

Examensarbete inom högskoleingenjörsprogrammet Datateknik

Johanna Johansson Hallberg
Kevin Östman

INSTITUTIONEN FÖR DATA- OCH INFORMATIONSTEKNIK

CHALMERS TEKNISKA HÖGSKOLA
GÖTEBORGS UNIVERSITET
Göteborg, Sverige 2022
www.chalmers.se

Autonom körning i realtid
Jämförelse av olika artificiella neurala nätverk

© Johanna Johansson Hallberg och Kevin Östman

Handledare: Sakib Sisteek, Chalmers Tekniska Högskola
Examinator: Jonas Duregård, Chalmers Tekniska Högskola

Kandidatarbete 2022
Institutionen för Data- och Informationsteknik
Chalmers Tekniska Högskola
SE-412 96 Göteborg
Sverige
Telefon + 46 (0)31-772 1000

Göteborg, Sverige 2022

Förord

Ett stort tack till vår handledare Alex Darborg på Knightec för stöttning och uppmuntran genom hela projektet. Vi vill även rikta ett tack till Joakim Brandt på Knightec för värdefulla synpunkter kring rapporten. Sist men inte minst vill vi rikta ett tack till vår handledare Sakib Sistek på Chalmers för hans kloka råd.

Sammanfattning

Artificiell intelligens gör utvecklingen av autonom körning både effektiv och simpel. Att med hjälp av en kamera samla in data har visat sig ge tillräckligt mycket information för att en bil självständigt ska kunna följa en markerad linje. Då artificiell intelligens har ett brett användningsområde uppstår det problem om att veta vilka artificiella nätverk som passar bäst till ändamålet. Syftet med det här projektet var att analysera hur olika nätverk och datamängder påverkar körningen för att slutligen välja ut den kombination som bäst kunde köra runt en markerad bana på kortast tränings tid. Bilen som användes var en JetRacer utrustad med en NVIDIA Jetson Nano enkorts dator. De olika nätverken tränades på bilder tagna ifrån bilens kamera för att sedan styra JetRacerns gaspådrag samt svängaxel. Då körningen skedde i realtid var det viktigt att nätverket kördes lokalt på enkorts datorn för att minimera risken av fördröjningar då data skickas mellan enheter. Efter att ha jämfört fyra olika nätverk var det ResNet-18 som var mest effektiv på minst mängd data och klarade därför att köra framåt, vänster, höger samt stanna vid hinder runt en bana. Detta arbete utfördes som ett komplementär projekt åt företaget Knightecs tidigare examensarbete inom utveckling av självkörande fordon. Som en förbättring för framtida projekt hade det varit intressant att utrusta bilen med en hastighetssensor som möjliggör körning i låga hastigheter.

Abstract

Artificial intelligence makes the development of autonomous driving both effective and simple. Using a camera to collect images has shown to give enough relevant information for a car to be able to independently follow a marked line. Because of artificial intelligence's broad area of usage, problems of knowing which artificial network to use for each purpose can arise. The purpose of this project is to analyze how different networks and datasets affect the car's driving ability in order to highlight the combination that most efficiently drove around a marked track in the least amount of time to train it. The car that was used was a JetRacer equipped with an NVIDIA Jetson Nano single card computer. Images from the camera on top of the car were used to train different networks in order to control acceleration and steering of the JetRacer. When driving in real time, it is important that the networks run locally on the single card computer to minimize the risk of latency when data is sent between devices. After comparing four different networks, ResNet-18 was found to be the most efficient one in terms of data usage, hence it managed to drive forwards, left, right and stop before hitting an obstacle. This degree project report was carried out as complementary to the company Knightec's previous work in the development of autonomous vehicles. A suggestion for future projects is adding a speed sensor, allowing better driving at lower speeds.

Terminologi och förkortningar

Dessa presenteras med kursiv text i rapporten.

Etcher	En programvara vilket används för att formatera samt installera mjukvara på SD-kort.
JetCard	En SD-korts avbild av Ubuntu förinstallerad med programvara specifikt för AI utveckling från NVIDIA.
SSH	Secure shell (SSH) är ett nätverksprotokoll vilket möjliggör en säker uppkoppling till andra datorer.
Epok	Varje gång en modell bearbetar all data i den datamängd som som tillhandahålls.

Innehållsförteckning

1 Inledning	1
1.1 Bakgrund	1
1.2 Syfte	1
1.3 Mål	1
1.4 Avgränsningar	1
2 Teknisk bakgrund	2
2.1 Artificiell intelligens och maskininlärning	2
2.1.1 Artificiellt neuralt nätverk	2
2.1.2 Överanpassning och underanpassning	3
2.1.3 Falttningsnätverk (CNN)	3
2.1.3 AlexNet	4
2.1.4 SqueezeNet	4
2.1.5 ResNet	4
2.1.6 DenseNet	4
2.2 Klassificering och regression	4
2.3 Utvärdering	5
2.3.1 Noggrannhet	5
2.3.2 Precision, recall och F1-score	5
2.3 Bild-Augmentering	6
2.4 Bibliotek	7
2.5 Jupyter Notebook & Jupyter Lab	7
2.6 Jetson Nano	7
2.6.1 JetRacer	8
2.7 Självkörande bilar	8
3 Metod och genomförande	9
3.1 Sätta upp JetRacern	9
3.2 Datainsamling	9
3.3 Datasetens struktur	10
3.4 För-processering av datan	10
3.5 Utveckling av modeller	11
3.6 Utvärdering av modeller	12
3.7 Testkörning	12
4 Resultat	14
4.1 Jämförelse av modellerna	14
4.2 Jämförelse av körning	16

5 Diskussion	17
5.1 Etiska aspekter	19
5.2 Ekologiska aspekter	19
6 Slutsats	20
6.1 Vidare forskning	20
Källförteckning	21

1 Inledning

Självkörande bilar blir allt mer vanliga i dagens bilindustri. Genom att utveckla AI Algoritmer som inte beror på mänsklig interaktion kan man minska risken för skador i trafiken. Detta då man kan eliminera faktorer såsom trötthet, alkohol och aggressivitet i samband med körning [1].

1.1 Bakgrund

Knightec är ett konsultföretag etablerat i bland annat Sundsvall, Stockholm och Göteborg. Företaget erbjuder en mängd tekniska tjänster inom bland annat Data Science, Machine Intelligence och Software Engineering. Företaget har idag, sedan tidigare examensarbete, en autonom AI-baserad radiobil som kan ta sig runt på en matta med tydligt utmarkerade kanter och vill i år vidareutveckla den [2]. Då stora dataset är kostsamma är det även viktigt att utvärdera effektiviteten hos olika algoritmer med minimal data för att uppnå en ekologiskt hållbar mjukvara.

1.2 Syfte

Idén är att utgå ifrån den redan befintliga radiobilen och vidareutveckla den till att klara av mer avancerade uppgifter såsom att ta sig runt en tejpade bana, identifiera hinder samt ändra hastighet. Då hårdvaran är prestandabegränsad är det viktigt att kunna utnyttja ett så effektivt artificiellt neuralt nätverk som möjligt. I detta projektet kommer därför olika nätverk samt olika mängd data att jämföras.

1.3 Mål

För att kunna uppnå syftet med projektet så kommer följande punkter att genomföras:

- Flera artificiella neurala nätverk skall tränas med insamlade bilder från bilens inbyggda kamera.
- Den befintliga radiobilen skall utvecklas till att kunna ta sig runt via tejpade linjer samt kunna bromsa in vid hinder.
- Flera dataset skall konstrueras i olika storlekar.
- Olika neurala nätverk skall utvärderas för att jämföra deras noggrannhet samt för att kunna jämföra hur många/få bilder det krävs för att uppnå önskat resultat.
- Bilens körförmåga skall utvärderas för att undersöka om de olika neurala nätverkens noggrannhet påverkar körningen.

1.4 Avgränsningar

Hårdvaran som används i detta projektet förbereddes under det föregående examensarbetet. För att möjliggöra att projektet mål ska kunna utföras så effektivt som möjligt så kommer vissa avgränsningar att göras:

- Utökning av hårdvara kommer inte att ingå i detta projektet.
- Beräkningar under aktiv körning kommer endast att ske lokalt på enhet, inte på molnet.

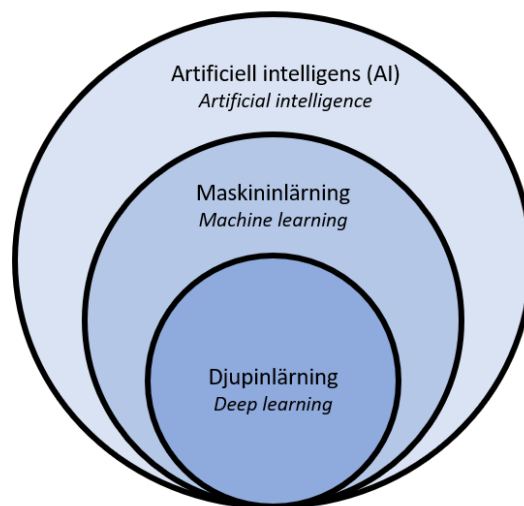
2 Teknisk bakgrund

Här presenteras de olika tekniker som har använts i projektet. Kapitlet börjar med att först beskriva vad artificiell intelligens och maskininlärning är. Därefter presenteras begreppen artificiellt neuralt nätverk, överanpassning, underanpassning samt falttningsnätverk. Efter detta presenteras de nätverk som använts i detta projekt samt olika verktyg.

2.1 Artificiell intelligens och maskininlärning

Artificiell intelligens (AI) är förmågan hos datorprogram att efterlikna människans och andra djurs naturliga intelligens. Värdefulla förmågor att kunna efterlikna är till exempel förmågan att lösa problem, lära sig av tidigare erfarenheter samt att kunna planera en sekvens av handlingar [3]. Inom tekniska termer refereras ofta AI algoritmer till att vara en modell.

Maskininlärning är en gren inom AI där fokuset ligger på att modellen ska kunna lära sig från data och därefter kunna ta beslut när den presenteras för ny data. Maskininlärning använder algoritmer för att kunna hitta och identifiera mönster och egenskaper i data [4]. Det går ut på att träna ett datorprogram till att klara en viss uppgift som till exempel att kunna identifiera handskrivna siffror eller att kunna urskilja vad som är ett fordon i trafiken. Inom maskininlärning används ofta artificiella neurala nätverk. Djupinlärning är en undergrupp till maskininlärning och består av artificiella neurala nätverk med tre eller fler lager. Djupinlärning är det som kommer att användas i detta projekt. I figur 1 nedan illustreras förhållandet mellan artificiell intelligens, maskininlärning samt djupinlärning.

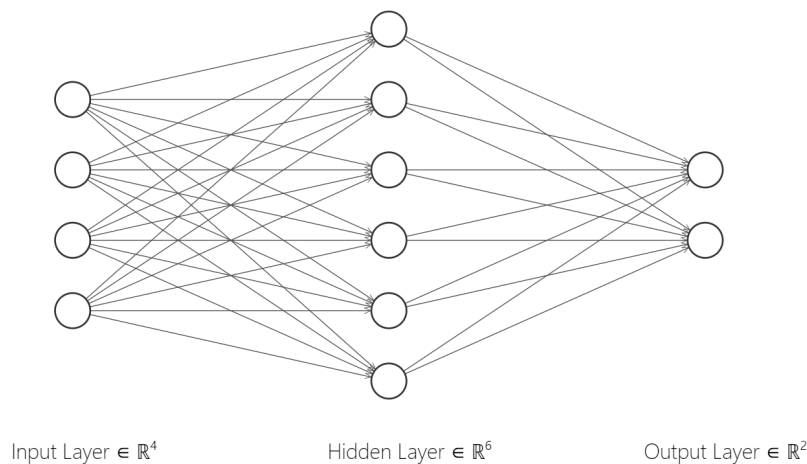


Figur 1. Förhållandet mellan AI, maskininlärning och djupinlärning.

2.1.1 Artificiellt neuralt nätverk

Artificiellt neuralt nätverk, även kallat artificiellt neuronät, består av sammankopplade noder i form av ett lager med indata, ett eller flera dolda lager samt ett lager för utdata [5]. Det är hur ett nätverk byggs upp som skapar dess unika egenskaper. Mängden lager samt kombinationen av olika slags lager bidrar till hur bra nätverket blir för dess ändamål. I figur 2 nedan visualiseras ett nätverk där lagrena är helt sammankopplade (*fully-connected layer på Engelska*), har ett dolt lager samt två olika kategorier som datan kan klassificeras till. Utdatan

från ett lager används som indata till varje nod i nästa lager. Inuti varje nod sker beräkningar och därefter skickas resultatet som indata till nästa lager eller som svar i form av utdata.



Figur 2. Artificiellt neuralt nätverk med två möjliga klassificeringar för utdata.

Likt människor behöver även ett neuralt nätverk tränas upp för att anpassas efter den ålagda uppgiften. Träningen sker genom att presentera data och låta nätverket gissa utifrån bestämda slutsatser för att sedan jämföra resultatet med facit. Efter att ett nätverk är skapat och tränat på datan har man skapat en modell.

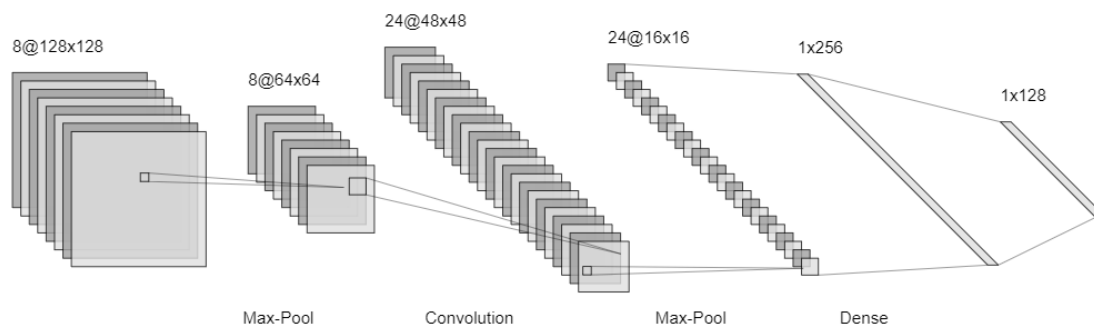
2.1.2 Överanpassning och underanpassning

Överanpassning (*overfitting på Engelska*) är ett koncept inom datavetenskap vilket innebär att modellen allt för precist har anpassat sig till datan den är tränad på. Detta medför att modellen presterar bra på träningsdatan men sämre på testdatan. Det finns flera olika orsaker till överanpassning men några vanliga orsaker är att datasetet innehåller för lite data eller att modellen har tränats för många gånger och därmed blivit presenterad för samma data onödigt många gånger. En annan orsak som kan leda till överanpassning är att man valt en modell som är för komplex för det dataset man har vilket kan resultera i att den hittar icke relevanta mönster [6].

Underanpassning (*underfitting på Engelska*) är ett koncept inom datavetenskap och betyder att modellen presterar undermåligt både på träningsdatan och testdatan. En orsak som kan leda till underanpassning är att modellen har tränats för få gånger för att kunna hitta relevanta mönster. En annan orsak som kan leda till underanpassning är att man valt en modell som är alldeles för simpel för det dataset man har vilket därför kan resultera i att den missar relevanta mönster [7].

2.1.3 Faltningsnätverk (CNN)

Faltningsnätverk (*Convolutional Neural Network på Engelska*) är ett neuralt nätverk som är väldigt bra lämpat åt analysering av bilder [8]. Ett CNN består av ett flertal lager som kallas för convolutional, pooling samt dense (se figur 3). Dessa lager syftar till att identifiera mönster bland varje input och i kombination med varandra skapar olika nätverk. Det finns ett flertal olika nätverk vars syfte och resultat varierar beroende på vilken uppgift som skall utföras.



Figur 3. En visuell representation av hur ett CNN kan se ut.

2.1.3 AlexNet

AlexNet är ett CNN vilket bildades 2012 och vann även det året ImageNets tävling Large Scale Visual Recognition Challenge (ILSVRC) [9 (sida 11)]. AlexNet består av åtta stycken lager varav de fem första lagren är convolutional lager och de tre sista är fully-connected lager. AlexNet tillåter att hälften av modellens noder tränas på en GPU och den andra hälften på en annan GPU vilket medför att tränings tiden minskar.

2.1.4 SqueezeNet

SqueezeNet är ett CNN likt AlexNet dock med femtio gånger färre parametrar. Det som skiljer SqueezeNet från andra nätverk hur litet och kompakt det är, där nätverket består av lager som först minskar parametrarna för att sedan expandera dem igen [10].

2.1.5 ResNet

ResNet är ett CNN och namnet kommer ifrån det Engelska namnet "Residual Network". I ett traditionellt neuralt nätverk hanteras utdatan från ett lager som indata till nästa. I ett ResNet är flera lager sammansatta i block, och samtidigt som utdatan från föregående lager hanteras som indata till nästa så används den även som indata till nästa block [9 (sida 12)].

De vanligaste ResNet arkitekturerna är ResNet-18, ResNet-34, ResNet-50, ResNet-101 samt ResNet-152 där siffran står för antal lager som nätverket består av.

2.1.6 DenseNet

DenseNet (*Dense Convolutional network*) är ett CNN där utdatan från varje lager är kopplat till alla efterföljande lager i ett Dense-block [9 (sida 13)]. DenseNet presterar bra vid klassificering men tar längre tid att träna och mer minne då det innehåller många lager. De vanligaste DenseNet arkitekturerna är DenseNet-121, DenseNet-169 och DenseNet-201 där siffran står för antal lager.

2.2 Klassificering och regression

Klassificering och regression är två olika metoder som en AI modell kan nyttja när den skall utvärdera ett resultat. Klassificering innebär att modellen urskiljer diskreta klasser utifrån en förutbestämt mängd för att kunna kategorisera varje utvärdering. Regression skiljer sig från

klassificering då det låter modellen urskilja värden inom ett kontinuerligt intervall och tillåter modellen att utvärdera en kvantitet [11].

2.3 Utvärdering

För att kunna avgöra hur bra en modell presterar finns det olika metoder som, med hjälp av matematiska formler, ger ett jämförelsevärde. Beroende på vad för typ av modell man har och hur datasetet är utformat finns det bättre lämpade utvärderingsmetoder.

2.3.1 Noggrannhet

Noggrannhet beräknas genom att man tar antalet korrekt klassificerade dividerat på den totala mängden, vilket ger den procentuella andelen korrekta klassificeringar inom datasetet. I figur 4 nedan visas ett exempel på beräkning av noggrannhet med fyra stycken klasser.

$$\text{Noggrannhet} = \frac{10 + 10 + 10 + 8}{40} = 0.95$$

Verklig klass	A	10	0	0	0
	B	0	10	0	0
	C	0	0	10	0
	D	1	0	1	8
		A	B	C	D
		Förutspädd klass			

Figur 4. Ekvation samt matris för att tydliggöra hur noggrannhet beräknas.

I de fall då datasetet är balanserat, alltså jämt fördelat mellan de olika klasserna så är noggrannheten ett bra verktyg för att mäta hur modellen presterar. I de fall då datasetet är obalanserat och klasserna inte är jämnt fördelade så kan noggrannhet vara ett bristfälligt verktyg att använda [12]. Ett exempel är ifall man har 100 bilder där 90 utav dessa är en katt och 10 st inte är en katt och låt säga att modellen gissar rätt på alla katter men den gissar också att alla icke-katter är katter. I detta fallet har modellen gissat rätt på 90 bilder och fel på 10 bilder vilket hade gett ett värde på $90/100 = 0.90$. En noggrannhet på 90% kan tolkas som ett bra resultat, men i detta fallet gissar modellen fel på allt som inte är katter.

2.3.2 Precision, recall och F1-score

Ett sätt för att motverka problemet som uppstår vid obalanserade dataset är att mäta hur bra en modell presterar både när modellen klassificerar korrekt, men också när modellen klassificerar felaktigt. På så sätt kan man mer noggrant analysera klasserna var för sig. De olika scenarier man fokuserar på benämns oftast på Engelska och kallas True-Positives (TP), False-Positives (FP), True-Negatives (TN) samt False-Negatives (FN). True-Positives och True-Negatives betyder att modellens förutsägelser var korrekta. False-Negatives och False-Positives betyder att modellens förutsägelser var felaktiga. I figur 5 nedan illustreras de olika scenarier som beskrivits innan.

Verklig klass	Positive	True-Positives TP	False-Negatives FN
	Negative	False-Positives FP	True-Negatives TN
		Positive	Negative
		Förutspådd klass	

Figur 5. En matris som visualiserar TP, FP, TN samt FN.

Med hjälp utav dessa värden kan man räkna ut precision, recall samt F1-score.

Precision används för att beräkna hur stor andel av de förutspådda positiva som klassificerades korrekt. Detta beräknas med formeln:

$$Precision = \frac{TP}{TP + FP} \quad (2.3.2.1)$$

Recall används för att beräkna hur stor andel av de verkligt positiva som klassificerades korrekt. Detta beräknas med formeln:

$$Recall = \frac{TP}{TP + FN} \quad (2.3.2.2)$$

F1-score är ett mått som ger en indikation på hur stor andel False-Negatives och False-Positives, med andra ord felklassificeringar, som modellen förutspår. Detta beräknas med formeln:

$$F1\ Score = \frac{2 \times Precision \times Recall}{Precision + Recall} \quad (2.3.2.3)$$

2.3 Bild-Augmentering

Augmentering är en teknik som används för att expandera och generalisera datasetet, motverka överanpassning samt öka modellens noggrannhet. Med hjälp utav det grundläggande datasetet kan augmentering generera ytterligare bilder genom att, till exempel, rotera, flippa, zooma eller förvränga dem samtidigt som bildernas klassificering fortfarande är densamma [13].

2.4 Bibliotek

PyTorch är ett öppet källkodsramverk för maskininlärning utvecklad av Meta Platforms, Inc. Inom detta ramverk kan man enkelt och smidigt importera eller skapa egna neurala nätverk och är en plattform för djupinlärning som levererar en hög flexibilitet och snabbhet [14].

2.5 Jupyter Notebook & Jupyter Lab

I en Jupyter Notebook kan man skriva och spara kod i en webbaserad miljö. Jupyter Notebook fungerar som en anteckningsbok indelad i olika boxar. Inuti en box kan man till exempel skriva dokumentation, exekvera Python kod samt måla ut grafer. Då en Jupyter Notebook är uppdelad i olika boxar kan man även köra varje sådan individuellt och lagra variabler lokalt under körning.

Jupyter lab är den senaste webbaserade utvecklingsmiljön för att hantera Jupyter notebooks, kod, samt data. Jupyter Lab tillhandahåller en snabb och överskådlig vy för att enkelt kunna hantera flera Jupyter Notebooks på samma gång [15].

2.6 Jetson Nano

Jetson Nano är en enkorts dator tillverkad av NVIDIA och är primärt ägnad åt embedded utveckling eller AI applikationer. Till strömförsörjning finns det antingen en 5v DC port eller en Micro-usb port. Utöver detta har datorn även ett MicroSD uttag för att kunna lagra os och filer. Jetson Nano har en kraftfull GPU vilket möjliggör utveckling av större och modernare tekniker som bl.a AI [16]. Hårdvaru-specifikationerna kan ses i tabell 1 nedan.

Tabell 1. Hårdvaru-specifikationer.

GPU	NVIDIA Maxwell arkitektur med 128 NVIDIA CUDA® kärnor
CPU	Quad-core ARM Cortex-A57 MPCore processor
Minne	4 GB 64-bit LPDDR4, 1600MHz 25.6 GB/s
Lagring	16 GB eMMC 5.1 samt minneskortsläsare
USB	4x USB 3.0, USB 2.0 Micro-B
Display	HDMI 2.0 samt eDisplayPort 1.4
Kamera- modul	2x CSI-2

2.6.1 JetRacer

JetRacer är en komplettering till Jetson Nano i form av en bil som omkapslar datorn och utökar dess funktionalitet. Det finns olika modeller på marknaden som var och en är designade olika och i detta projekt används en JetRacer producerad av Waveshare (se figur 6). Denna JetRacer ger datorn möjlighet att kunna styra en motor och ett styrservo, få tillgång till en kamera samt att kunna köras på externa batterier. JetRacern tillhandahåller även en display som visar batterinivå samt den aktuella ip adressen för Jetson Nanon. [17]



Figur 6. JetRacer utrustad med en Jetson Nano.

2.7 Självkörande bilar

Inom området självkörande bilar kan man referera till 5 olika grader om hur autonom bilens körning är [18]. Alla grader varierar i både hård- och mjukvara där grad 0 är en manuellt styrd bil och grad 5 är en självstyrande bil utan mänsklig interaktion. Två av dessa autonoma system är uppbyggda på olika sätt men tjänar samma syfte. Det första systemet utnyttjar diverse teknologi såsom kamera, LIDAR, och radar system som i samverkan med varandra arbetar för att uppnå ett självkörande system. Det andra systemet nyttjar neurala nätverk som, med hjälp av bilder, fattar beslut vilket uppnår ett likvärdigt system. Det senare alternativet är både mer lönsamt och enklare då mindre kod samt hårdvara involveras, vilket lett till att neurala nätverk fått ett större fokus inom utvecklingen av självkörande bilar [19].

I det tidigare examensarbetet vilket detta projekt är en utveckling på användes en matta med markerade körlinjer för att träna ett neuralt nätverk och skapa en självkörande bil [1]. Resultatet från arbetet visar att det är möjligt att skapa en bil i miniformat som med hjälp av bildanalys från ett neuralt nätverk klarar att ta sig runt en matta.

3 Metod och genomförande

I detta kapitel redogörs de tillvägagångssätt och metoder som har använts under projektet.

Noggrannhet mättes på de modeller som använts tillsammans med tiden det tog att träna varje modell på olika stora dataset. De bästa kombinationer av modell och datamängd testades därefter på en bana för att avgöra praktisk duglighet.

3.1 Sätta upp JetRacern

För att komma igång med JetRacern behövdes Jetson Nano datorn formateras och omkonfigureras. Med hjälp av en minneskortsläsare kopplades SD-kortet som medföljde enkortsdatorn in på en separat laptop. Väl inkopplad i datorn användes programvaran *Etcher* för att formatera och installera *JetCard* på SD-kortet. Då Jetson Nano var av en senare modell var det viktigt att *JetCard* versionen stödde revision B01. Till en början konfigurerades enkortsdatorn med tangentbord och mus för att sedan övergå till att styra Jetson nano från en extern laptop. Detta gjordes genom att etablera en SSH uppkoppling mellan enkortsdatorn och den externa datorn.

Efter att operativsystemet initialiserats installerades nödvändigheter såsom kamera funktionalitet och Jetracerns funktioner. Detta genom att följa steg 5 i installationsguiden från Waveshares GitHub sida för JetRacern [20]. För att kunna styra motorer och däck för just Waveshare modellen som användes i detta projekt krävdes det att i steg 5.3 i installationsguiden byta ut "NVIDIA-AI-IOT" mot "waveshare".

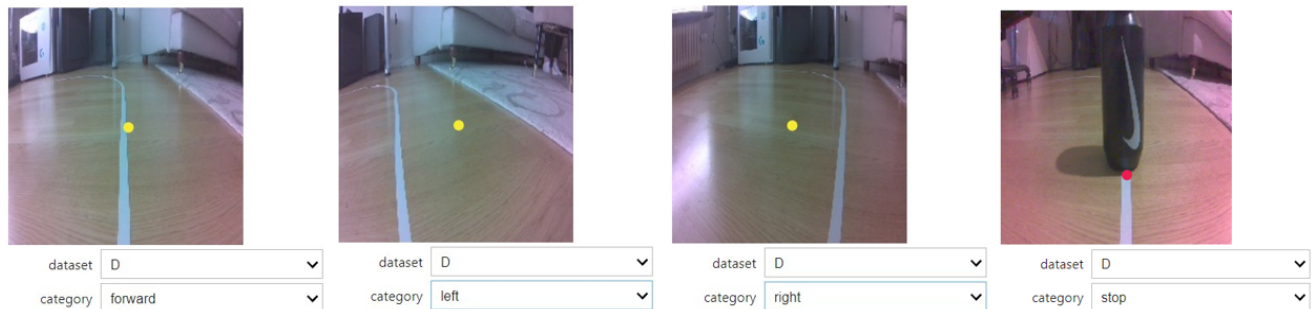
Efter att all installation var slutförd behövdes Jupyter Lab initialiseras vid varje uppstart av Jetson Nano för att kunna köra Python kod genom Jupyter notebooks. För att undvika behovet av att exekvera kod direkt på Jetson Nano öppnades även Jupyter Lab upp lokalt för det nätverk den var ansluten till. Detta gjordes genom att exekvera koden "Jupyter Lab --ip 0.0.0.0" i terminalen. För att ansluta till Jupyter Lab kunde man därefter besöka enkortsdatorns ip adress följt av porten 8888 i webbläsaren.

3.2 Datainsamling

För att kunna träna de CNN nätverk som användes behövdes det samlas in data i form av bilder. Då JetRacern har en inbyggd kamera placerad längst fram på bilen var det den som användes för att samla in all data. Med en Jupyter Notebook skapad av NVIDIA kunde man etablera en direkt åtkomst till kameran och ta bilder. Innan insamling av bilder krävdes det att koden modifierades för att passa projektets ändamål. Efter att fyra olika klassificeringar implementerades, fram, stopp, höger och vänster, kunde man slutligen klassificera varje bild med föredragen klass och skapa ett dataset.

Utgångspunkten för datainsamling skedde på en utmarkerad bana av tejp där det togs bilder i olika platser och vinklar för att kategorisera varje bild utifrån hur bilen skulle uppföra sig vid liknande scenarion (se figur 7). En gul utmarkerad prick 30 cm framför bilen användes som utgångspunkt för att kunna klassificera vilket håll bilen borde köra vid alternativen fram, vänster och höger. Vid stopp användes en cerise prick 20 cm framför bilen. Varje bild var

färgad, hade en 224x224 storlek och placerades automatiskt i en mapp vars namn motsvarade den tilldelade kategorin. Då alla bilder hade tagits bestod datasetet av totalt 500 bilder jämt fördelat över de olika kategorierna.

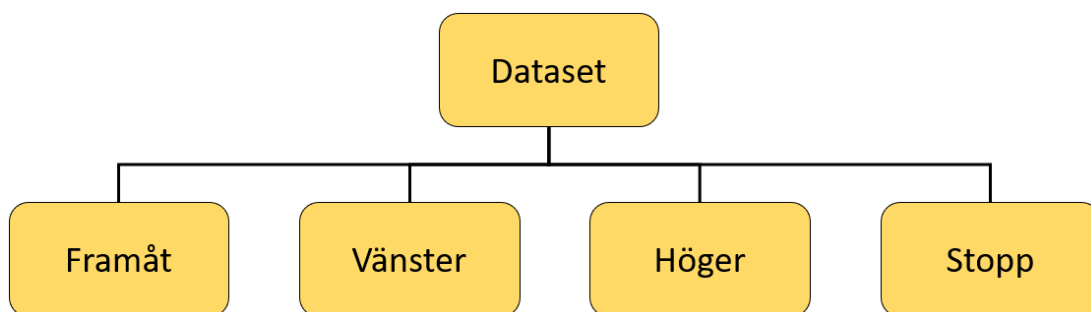


Figur 7. Klassificering av bilderna fram, vänster, höger och stopp.

För att kunna analysera hur många bilder som krävs för ett bra resultat samt för att kunna undvika överanpassning så utökades datasetet med hjälp av augmentation. I detta fallet valdes en svag rotation upp till 10 grader åt höger eller vänster. Utökningen justerade varje bild i datasetet och sparade dessa tillsammans med originalbilden. Med hjälp av detta skapades sju stycken nya dataset utöver originalet, dessa dataset innehöll 2500, 3700, 4500, 5100, 6100, 7300 samt 8500 bilder. De totalt åtta dataseten användes sedan till att kunna jämföra hur många/få bilder det krävs för att uppnå önskat resultat.

3.3 Datasetens struktur

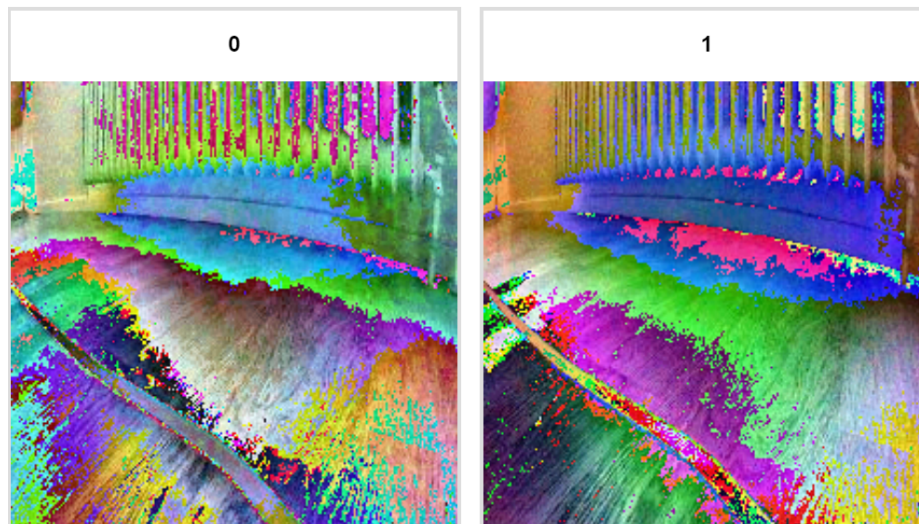
Alla dataset bestod av olika många bilder, men strukturen var likadan. Bilderna låg i mappar för varje kategori vilka var "Framåt", "Vänster", "Höger", och "Stopp". Alla de fyra olika kategori-mapparna låg under en gemensam mapp (se figur 8).



Figur 8. Visualisering av datasetets struktur.

3.4 För-processering av datan

När bilder hämtats från datasetet appliceras det en slumpad filtrering av bildens ljusstyrka, kontrast, mättnad och nyans på en skala mellan 0 % till 20 %. Bilderna normaliseras även för att intensifiera partier av bilder och öka chansen att hitta olika mönster. Då filtreringen sker slumpartat varje gång en bild hämtas från datasetet bidrar det till att samma bild kan visa olika mönster demonstrerat nedan i figur 9. Detta leder till att samma bild kan återanvändas för att träna modellerna utan att de överanpassas efter varje *epok*.



Figur 8. Slumpvald applicering av filter på identiska bilder

För att inte evaluera modeller på samma data som den tränar på var det viktigt att dela upp data inom träning och test. Med hjälp av PyTorch delades därför alla datasetens bilder upp i 80 % träning och 20 % test.

3.5 Utveckling av modeller

När all data som behövs är insamlad och för-processad är nästa steg att skapa modellen. En modell är ett nätverk som har blivit tränat på data, i detta fallet bilder. Då ett av projektets mål var att jämföra och utvärdera olika modeller har fyra stycken modeller utvecklats. De nätverk som valts till att utveckla modellerna var AlexNet, SqueezeNet, DenseNet-121 samt Resnet-18 då dessa hade provkörts i det tidigare examensarbetet på Knightec men inte utvärderats.

För att komma igång med utvecklingen av modellerna användes de Jupyter Notebook filer som är tillhandahållna via NVIDIAs GitHub repository som en bas för att importera och träna färdiga nätverk från PyTorch (se figur 9). Då träning av modeller är en resurskrävande process valdes det att göra detta på en extern dator, spara modellen lokalt på hårddisken och sedan flytta över modellen till JetRacern. Under träning av modellerna kördes även en evaluering efter varje *epok* för att kunna spara värdet på modellens noggrannhet under hela utvecklingsprocessen.

```

# ALEXNET
#model = torchvision.models.alexnet(pretrained=True)
#model.classifier[-1] = torch.nn.Linear(4096, len(dataset.categories))

# SQUEEZENET
#model = torchvision.models.squeezenet1_1(pretrained=True)
#model.classifier[1] = torch.nn.Conv2d(512, len(dataset.categories), kernel_size=1)
#model.num_classes = len(dataset.categories)

# DENSENET 121
#model = torchvision.models.densenet121(pretrained=True)
#model.num_classes = len(dataset.categories)

# RESNET 18
model = torchvision.models.resnet18(pretrained=True)
model.fc = torch.nn.Linear(512, len(dataset.categories))

```

Figur 9. Kod som importerar nätverk där det aktuella nätverket är Resnet 18

3.6 Utvärdering av modeller

I detta projektet bestod datasetet av fyra balanserade klasser och därför var noggrannhet en bra metod för att utvärdera modellerna. I de fall då man får en låg noggrannhet kan precision, recall och F1-score användas för att utvärdera klasserna var för sig. Man räknar då ut dessa för varje separat klass och kan utifrån det få en uppfattning om vilken/vilka klasser som modellen ofta förutspår inkorrekt. För att kunna ta ett beslut kring vilket nätverk som presterade bäst beslutades det att jämföra träningstid samt storleken på dataseten då noggrannheten uppmättes till minst 95 %.

3.7 Testkörning

För att utvärdera hur en modell presterade på verklig data behövde strömmen av bilder från JetRacers kamera matas in till modellen i en kontinuerlig loop (se figur 10). Vid varje iteration strömmas en ny bild in till modellen vilken gör en gissning av hur den aktuella bilden skall klassificeras inom ett heltalsintervall mellan 0 och 3. Till följd av hur modellen klassificerat bilden ändras bilens fysiska egenskaper såsom styrservons rotation och gaspådrag. Beroende på modellens tidigare klassificering påverkas även gaspådraget annorlunda. Bilen test-kördes på en utmarkerad bana med vit tejp på ett parkettgolv.

```

prevIndex = 0

while True:
    clear_output(wait=True)
    image = camera.read()
    image = preprocess(image).half()
    output = model_trt(image).detach().cpu().numpy().flatten()
    index = np.argmax(output, axis=0)

    if index == 0: # Vänster
        car.steering = 0.8
        if prevIndex == 3:
            car.throttle = 0.8
            time.sleep(0.05)
        else:
            car.throttle = 0.4

    elif index == 1: # Framåt
        car.steering = 0
        if prevIndex == 3:
            car.throttle = 0.8
            time.sleep(0.05)
        else:
            car.throttle = 0.4

    elif index == 2: # Höger
        car.steering = -0.9
        if prevIndex == 3:
            car.throttle = 0.8
            time.sleep(0.05)
        else:
            car.throttle = 0.47

    else: # Stopp
        car.throttle = 0

    prevIndex = index

```

Figur 10. En while-loop som bestämmer hur bilen skall köra vid modellens gissningar.

4 Resultat

I detta kapitel redogörs de resultat som uppnått under projektets gång. Först presenteras en jämförelse av modellerna gällande tid och noggrannhet. Därefter presenteras en jämförelse av bilens körförmåga.

4.1 Jämförelse av modellerna

För att jämföra hur modellerna presterade på de åtta dataseten gjordes jämförelser av tiden det tog att träna modellerna samt vilken noggrannhet modellerna fick. Alla modeller tränades under 30 *epoker*. Resultaten kan ses i tabell 2 och 3 nedan.

Tabell 2. Noggrannhet samt träningstid för AlexNet och SqueezeNet.

Storlek på dataset	AlexNet Noggrannhet	AlexNet Träningstid	SqueezeNet Noggrannhet	SqueezeNet Träningstid
500	22 %	26 s	48 %	16 s
2500	21 %	94 s	99 %	111 s
3700	23 %	137 s	98 %	136 s
4500	25 %	145 s	100 %	166 s
5100	26 %	217 s	98 %	199 s
6100	25.5 %	267 s	100 %	244 s
7300	24 %	312 s	99 %	256 s
8500	24.5 %	392 s	95 %	307 s

Tabell 3. Noggrannhet samt träningstid för ResNet-18 och DenseNet-121.

Storlek på dataset	ResNet-18 Noggrannhet	ResNet-18 Träningstid	DenseNet-121 Noggrannhet	DenseNet-121 Träningstid
500	90 %	35 s	97 %	106 s
2500	100 %	135 s	100 %	452 s
3700	100 %	241 s	100 %	677 s
4500	100 %	263 s	100 %	789 s
5100	100 %	297 s	100 %	898 s
6100	100 %	349 s	100 %	1059 s
7300	100 %	414 s	100 %	1296 s
8500	100 %	480 s	99 %	1538 s

Resultatet som redovisas i tabell 2 visar att AlexNet endast kommer upp till en noggrannhet på 26 % vilket inte är ett önskvärt resultat då man strävar efter en så hög noggrannhet som möjligt. Resterande resultat är därför baserade på SqueezeNet, ResNet-18 och DenseNet-121.

De tre återstående nätverken uppnår en hög noggrannhet redan vid en låg mängd data. Vi valde att jämföra när modellerna uppnår en noggrannhet över 95 % och resultaten kan då sammanställas i tabell 4 nedan.

Tabell 4. Storlek på dataset, träningstid samt noggrannhet.

Nätverk	Storlek på dataset	Tränings- tid	Noggrannhet
SqueezeNet	2500	111 s	99 %
ResNet-18	2500	135 s	100 %
DenseNet-121	500	106 s	97 %

4.2 Jämförelse av körning

Jämförelse av körning gjordes med modellerna SqueezeNet, ResNet-18 samt DenseNet-121. Modellerna test-kördes två varv både med- och moturs på banan som visas i figur 11 nedan. Bilen klarade av att köra runt banan moturs utan att hamna utanför med alla tre modellerna, dock körde bilen utanför banan med modellerna SqueezeNet samt DenseNet-121 vid medurs varv. Med samtliga modeller klarade bilen av att stanna vid hinder. Resultatet från test-körningarna blir att ResNet-18 är bäst lämpat för hur banan är utformad i detta projektet då bilen klarade av både med- och moturs samt att stanna vid hinder.



Figur 11. Den utmarkerade bana som använts under projektet.

5 Diskussion

Under arbetets gång har datasets struktur korrigerats för att kunna prestera så bra som möjligt. Ett problem som uppstod till en början var att bilen ofta hamnade utanför körbanan. Då banorna endast bestod utav en tejpbit var det återkommande fall då bilen förlorade tejpen ur sikte och var för dåligt tränad för att veta vad den skulle göra. Till en början motverkades detta problem genom att träna bilen till att backa tillbaka in i banan varje gång den förlorat tejpen ur sikte. Det märktes dock snabbt att bilen hade klarat majoriteten av svängarna där den förlorat tejpen ur sikte ifall den bara hade fortsatt att svänga. Ifall banan istället hade varit konstruerad med två tejpade linjer hade bilen haft mer information om svängar att förlita sig på. Vi valde därför att ta bort funktionaliteten av att backa och istället fokusera på att hålla hastigheten låg för att enkelt kunna ta sig runt svängar inom banan.

Eftersom bilens hastighet bestämdes genom hur hög kraft man gav motorerna var det inte fysiskt möjligt för bilen att kunna ta sig runt banan i den mest optimala hastigheten som hade varit önskvärd. Detta då den inte fick tillräckligt mycket kraft för att kunna motverka friktionen på hjulen vilket medförde att bilen fastnade i stillastående läge. En lösning till detta hade varit att man installerade hastighetssensorer på bilens däck för att mäta hjulens rotation per minut i realtid. Man hade då kunna ställa in en bestämd hastighet och programmera bilen till att ge mer kraft då hjulen går för långsamt, eventuellt mindre kraft då hjulen går för snabbt.

En annan bidragande faktor till hur bra bilen presterade ligger i hur bra datasetet är konstruerat. Den mänskliga faktorn har en stor benägenhet till att påverka vilka bilder som skall klassificeras som vad. Eftersom detta arbete utfördes av två studenter var det viktigt att utse enbart en av dessa till att konstruera datasetet för att inte blanda in motsägelsefulla uppfattningar kring hur bilen skall agera. Då vi även fick mäta om den prick som vi utgick ifrån när datasetet konstruerades kan vi inte vara säkra på att den låg på exakt samma position varje gång vi utökade datasetet.

En yttre faktor som bidrog till att bilen körde utanför banan eller svängde åt fel håll var solljuset. Solljuset bildade streck på golvet vilket bilen tolkade som linjer (se figur 12). För att undvika detta problem behövdes allt solljus hindras från att hamna på och bredvid körbanan.



Figur 12. Soljus på banan som påverkar körning

Vid jämförelse av körningen minimerades påverkan från solljuset genom att fönster täcktes för. De tre modellerna SqueezeNet, ResNet-18 samt DenseNet-121 tränades med deras respektive bästa förutsättningar och sedan testkördes bilen med alla de tre modellerna. Modellerna presterade ungefär likadant, vilket förklaras av att de alla uppnådde en hög noggrannhet.

Som presenterats i resultatet så körde ResNet-18 runt banan två varv både med- och moturs utan att köra utanför banan, medan SqueezeNet och DenseNet-121 körde utanför vid medurs när de kom till någon av svängarna. Att de två sistnämnda modellerna körde utanför kan bero på att det tar längre tid för modellerna från att en bild tas tills dess att modellerna förutspått vilken klassificering bilden bör få. Detta medför att när bilen kommer till en sväng så hinner bilen köra rakt fram och således missa tejen innan modellen har hunnit bestämma att bilen ska svänga, och när väl tejen är ur sikte så hittar inte bilen tillbaka till banan.

I framtida liknande projekt hade det varit intressant att skapa en mängd olika banor för att undersöka ifall banans form hade påverkat resultatet. Ifall svängarna hade varit mer skarpa så hade det funnits en möjlighet att alla tre modellerna hade missat svängarna. Ifall svängarna hade varit mindre skarpa så hade det funnits en möjlighet att alla tre modellerna hade klarat av svängarna och hållit sig kvar på banan.

Från tabell 2 i resultatet kan man notera att AlexNet uppvisade låg noggrannhet vilket medförde att det nätverket togs bort från resterande jämförelser. De tre övriga nätverken visade allihopa på en noggrannhet över 95 %, dock med varierande mängd data.

DenseNet-121 är det nätverk som redan vid det minsta datasetet kom upp i en hög noggrannhet. Dock kan man i tabell 3 se att träningstiden för DenseNet-121 ökar mycket när storleken på datasetet ökar, denna ökning är inte lika kraftig för de andra nätverken. Då storleken på datasetet var 8500 bilder så tog det ungefär 1500 s för modellen att träna, medan det tog ca 300-400 s för de andra nätverken. Ifall man hade utökat projektet och tillfört mycket fler bilder så hade DenseNet-121 blivit en tidskrävande modell.

5.1 Etiska aspekter

Autonom körning medför att den mänskliga faktorn i trafiken begränsas [1]. Ifall körningen inte längre påverkas av människan medför detta att det är upp till den artificiella intelligensen att utföra ställningstaganden under färd. Då en olycka skulle uppstå är det AI:n som väljer hur den skall agera i en situation där antingen passagerarna eller omvärlden skall prioriteras. Det kan därför vara upp till länder själva att ta etiska ställningstaganden som lägger grunden till hur partisk programvaran kommer att framstå [21].

Vidare kan det diskuteras om vilket etiskt dilemma som uppstår i samband med att man använder sig utav bilder som data. Då en bil med kamera skulle utsättas för en hackningsattack och övertas skulle inkräktare kunna få tag i bilder direkt från bilens kamera vilket skulle skapa integritetsproblem. Samhället behöver därför reglera detta problem med hjälp av lagar såsom GDPR för att säkerhetsställa för konsumenter att den data som samlas hanteras på ett sätt som inte skulle kunna orsaka skada [22].

5.2 Ekologiska aspekter

Ifall alla bilar skulle vara självkörande hade det varit möjligt att minska bränsleförbrukningen med upp till 90 %. Självkörande bilar hade kunnat leda till snabbare färd, lättare fordon, mindre tid spenderat på att parkera men även i sin tur mer resor. Då resor går både effektivare och snabbare kan det som konsekvens leda till att även resor och resvägen ökar, vilket i sin tur kan ha en motsatt effekt på miljön med en energiökning på 250 % [23].

6 Slutsats

Syftet med det här projektet var att vidareutveckla JetRacern från att enbart kunna köra runt på en matta med utmarkerade kanter till att kunna köra utefter en tejpade linje samt att kunna bromsa in vid hinder. Resultatet visar att JetRacern klarar, under rätt förhållanden såsom att minimera solljus, både att köra utefter en tejpade bana samt att kunna bromsa in vid olika hinder.

En jämförelse av fyra olika artificiella neurala nätverk har utförts och det som har jämförts är noggrannhet, träningstid samt mängden data. Resultatet som uppnåddes visade att av de fyra neurala nätverk som jämfördes så var DenseNet-121 det nätverk som kräver minst mängd data för att uppnå en hög noggrannhet medan ResNet-18 var det nätverk som presterade bäst på den bana som används i detta projekt.

Sammanfattningsvis har det här projektet visat att det är fullt möjligt att med hjälp av bilder, bild klassificering och artificiell intelligens skapa en självkörande bil som kan ta sig fram längs en tejpade linje samt stanna för hinder.

6.1 Vidare forskning

Inför framtida liknande projekt hade det varit intressant om hårdvaran utökades med hastighetssensorer för att kunna motverka problem som uppstår när bilen skall köras i låga hastigheter. Vidare hade det varit intressant att jämföra körningen på flera olika banor för att undersöka om de olika neurala nätverken ändrar beteende om banans utformning ändras.

Ett av de stora problemen under projektets gång har varit solljusets påverkan. I framtida projekt hade det kunnats göra en undersökning om man kan motverka problemen som uppstår under olika ljusförhållanden.

Källförteckning

- [1] P. Olsson, "Mänsklig faktor på väg ut ur trafiken," *logistikmagasinet*, vol. 3, s. 22, Sep. 2017. [Online]. Tillgänglig: <https://logistikmagasinet.prenly.com/p/logistikmagasinet/2017-09-01/a/mansklig-faktor-pa-va-g-ut-ur-trafiken/205/12977/991893>, Hämtad: 2022-05-23.
- [2] T. Bäckman, "Självkörande autonom bil i miniformat," examensarbete på kandidatnivå, Institutionen för data och informationsteknik, Chalmers tekniska högskola, Göteborg, Sverige, 2021. [Online]. Tillgänglig: <https://hdl.handle.net/20.500.12380/302347>
- [3] Umeå Universitet, "Vad är artificiell intelligens?," [Online]. Tillgänglig: <https://www.umu.se/forskning/var-forskning/fordjupa-dig/artificiell-intelligens/vad-ar-artificiell-intelligens/> (hämtad: 2022-03-29).
- [4] Microsoft Azure, "Vad är maskininlärning?," [Online]. Tillgänglig: <https://azure.microsoft.com/sv-se/overview/what-is-machine-learning-platform/#benefits> (hämtad: 2022-03-29).
- [5] IBM Cloud Education, "Neural Networks," 2020. [Online]. Tillgänglig: <https://www.ibm.com/cloud/learn/neural-networks> (hämtad: 2022-04-05).
- [6] IBM Cloud Education, "Overfitting," 2021. [Online]. Tillgänglig: <https://www.ibm.com/cloud/learn/overfitting> (hämtad: 2022-05-20).
- [7] IBM Cloud Education, "Underfitting," 2021. [Online]. Tillgänglig: <https://www.ibm.com/cloud/learn/underfitting> (hämtad: 2022-05-20).
- [8] IBM Cloud Education, "Convolutional Neural Networks," 2020. [Online]. Tillgänglig: <https://www.ibm.com/cloud/learn/convolutional-neural-networks> (hämtad: 2022-04-05).
- [9] Md Z. Alom et al., "The History Began from AlexNet: A Comprehensive Survey on Deep Learning Approaches", 2018, <https://doi.org/10.48550/arXiv.1803.01164>.
- [10] F. N. Iandola et al., "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size," 2016, <https://doi.org/10.48550/arXiv.1602.07360>.
- [11] J. Delua, "Supervised vs. Unsupervised Learning: What's the Difference?," *IBM*, [Online], Mar. 12, 2021. Tillgänglig: <https://www.ibm.com/cloud/blog/supervised-vs-unsupervised-learning> (hämtad: 2022-05-20).
- [12] Zack, "F1 Score vs. Accuracy: Which Should You Use?," *Statology*, [Online]. Sep. 08, 2021. Tillgänglig: <https://www.statology.org/f1-score-vs-accuracy/> (hämtad: 2022-05-23).
- [13] M. D. Bloice, C. Stocker, och A. Holzinger, "Augmentor: An Image Augmentation Library for Machine Learning," 2017, <https://doi.org/10.48550/arXiv.1708.04680>.

- [14] N. Ketkar, och J. Moolayil, "Introduction to PyTorch," i *Deep Learning with Python: Learn Best Practices of Deep Learning Models with PyTorch*, 2. uppl., Berkeley, CA, USA: Apress, 2021, kap. 2, ss. 27-91. [Online] Tillgänglig: <https://link.springer.com/content/pdf/10.1007/978-1-4842-5364-9.pdf>, Hämtad: 2022-05-17.
- [15] Project Jupyter, 2022. [Online]. Tillgänglig: <https://jupyter.org/> (hämtad: 2022-03-29).
- [16] NVIDIA, "Jetson Nano Developer Kit," 2022. [Online]. Tillgänglig: <https://developer.nvidia.com/embedded/jetson-nano-developer-kit> (hämtad: 2022-05-02).
- [17] Waveshare, "JetRacer AI Kit, AI Racing Robot Powered by Jetson Nano," [Online]. Tillgänglig: <https://www.waveshare.com/jetracer-ai-kit.htm> (hämtad: 2022-05-02).
- [18] K. Hyatt, C. Paukert, "Self-driving cars: A level-by-level explainer of autonomous vehicles," *CNET*, Mar, 29, 2018. [Online]. Tillgänglig: <https://www.cnet.com/roadshow/news/self-driving-car-guide-autonomous-explanation/>, Hämtad: 2022-06-12.
- [19] J. del Egio, L.M. Bergasa, E. Romera, C. Gómez Huélamo, J. Araluce, R. Berea, "Self-driving a Car in Simulation Through a CNN," i *WAF 2018: Advances in Physical Agents*, Madrid, Spanien, 2018, ss. 31-43. [Online]. Tillgänglig: https://link.springer.com/chapter/10.1007/978-3-319-99885-5_3, Hämtad: 2022-06-12.
- [20] GitHub, "Software Setup," 2019. [Online]. Tillgänglig: https://github.com/waveshare/jetracer/blob/master/docs/software_setup.md (hämtad: 2022-05-02).
- [21] J. Frljevic, "Tyskland har satt etiska riktlinjer för självkörande bilar," *Teknikens Värld*, Aug, 27, 2017. [Online]. Tillgänglig: <https://teknikensvarld.expressen.se/nyheter/bil-och-trafik/tyskland-har-satt-etiska-riktlinjer-for-sjalkvkorande-bilar-526869/>, Hämtad: 2022-05-23.
- [22] Integritetsskyddsmyndigheten, "The purposes and scope of GDPR," 2021. [Online]. Tillgänglig: <https://www.umu.se/forskning/var-forskning/fordjupa-dig/artificiell-intelligens/vad-ar-artificiell-intelligens/> (hämtad: 2022-05-23).
- [23] A. Brown, B. Repac, J. Gonder, "Autonomous Vehicles Have a Wide Range of Possible Energy Impacts," National Renewable Energy Laboratory, Stanford, CA, USA, NREL/PO-6A20-59210, 2013. [Online]. Tillgänglig: <https://www.nrel.gov/docs/fy13osti/59210.pdf>, Hämtad: 2022-05-23.



CHALMERS