Mia Fornes
Asbjørn Fiksdal Kallestad
Maria Lande

Bacheloroppgave

Deteksjon av biler gjennom en bomstasjon ved bruk av maskinlæring

Mai 2021

NTNU
Norges teknisk-naturvitenskapelige universitet.
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for datateknologi og informatikk

NTNU
Kunnskap for en bedre verden

Bacheloroppgave

2021

NTNU

Kunnskap for en bedre verden

Mia Fornes
Asbjørn Fiksdal Kallestad
Maria Lande

Deteksjon av biler gjennom en bomstasjon ved bruk av maskinlæring

Bacheloroppgave
Mai 2021

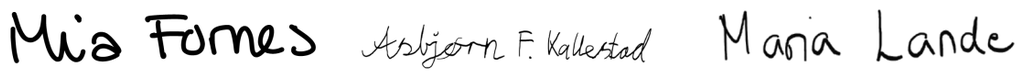
NTNU
Norges teknisk-naturvitenskapelige universitet.
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for datateknologi og informatikk

NTNU
Kunnskap for en bedre verden

Forord

Denne bacheloroppgaven er skrevet i forbindelse med studieretningen *Dataingeniør* ved Norges Teknisk-Naturvitenskapelige Universitet (NTNU) våren 2021, og er gjennomført i samarbeid med Q-Free ASA. Oppgaven utarbeidet av Q-Free ble ansett som svært relevant for tiden, og lot oss utforske interessante grener innenfor maskinlæring. Det har vært en lærerik prosess som har gitt oss en innsikt på arbeidslivet.

Vi ønsker å takke veileder Ole Christian Eidheim for hans veiledning og tilbakemeldinger i løpet av perioden. Vi ønsker også å takke Børge Godejord og Mark terBrugge fra Q-Free for behjelpeligheten og tilbakemeldingene underveis for å sikre kvaliteten på sluttproduktet. En stor takk rettes også til resten av Q-Frees team som gjennom hele prosjektperioden har vist stort engasjement. Videre ønsker vi å takke Idun for tilgang på nødvendige ressurser i forbindelse med trening av modeller. Til slutt ønsker vi å rette en takk til Lene Margrethe Pallesen for lesing av korrektur, samt venner og familie for støtten gitt underveis.



Mia Fornes Asbjørn Fiksdal Kallestad Maria Lande

Trondheim, 20.05.2021

Oppgavetekst

Oppgaven omhandler å bruke maskinlæring for å detektere biler i det de passerer gjennom en bomstasjon. Oppgaven fokuserer på deteksjon av personbiler da det var kun var tilgjengelig og tilstrekkelig data av denne typen kjøretøy. Målet er å finne gode alternativer til objektdeteksjonalgoritmer slik at pålitelig sporing av biler i sanntid kan være mulig. Det systemet Q-Free benytter seg av i dag for registrering av biler som passerer deres bomstasjoner krever at datakilder fra forskjellig tid og sted må kobles sammen med rett kjøretøy.

Den opprinnelige oppgaveteksten fokuserte på å detektere bildeler, for eksempel vindusrute, bilhjul og sidespeil, i tillegg til hele biler. Originalt var det også tenkt at løsningen skulle kunne fungere med Coral USB Accelerator [1] slik at resultatet kan brukes i bomringen. Det var også opprinnelig et større fokus på sporingsalgoritmer og optimalisering av dette. Original oppgavetekst kan ses i Vedlegg A.

Sammendrag

Denne oppgaven undersøker to alternativer til objektdetekteringsalgoritmer for deteksjon av biler under Q-Free sine bomstasjoner. Algoritmene som oppgaven omhandler er YOLO og SSD, som begge er regnet som ledende algoritmer innenfor objektdeteksjon. Algoritmene er trent og evaluert på Q-Free sitt datasett bestående av video tatt av vidvinkelkamera plassert på Q-Free sine bomstasjoner. To varianter av datasettet er testet for begge modellene, ett med og et uten tomme veier. Algoritmene sammenlignes opp mot hverandre med mål om å oppnå en mest mulig presis og effektiv deteksjon slik at deteksjonene kan brukes til sporing av biler ved passering av en bomstasjon. Oppgaven har kun sett på lettvektversjonene til YOLO og SSD for at kjøring på lettvekt maskinvare skal være mulig for framtidig implementasjon i Q-Free sitt sporingssystem.

Resultatene som diskuteres består av både evaluering av de trente modellene og visualisering av deteksjonene. Algoritmene oppnår gode resultater for detektering av biler under bomstasjonene, hvor de beste resultatene for både SSD og YOLO ligger mellom 98% og 100% for mAP og på 98% for F1 nøyaktigheten. De visuelle resultatene viser derimot at det fortsatt er tilfeller der modellene predikerer feil som kan påvirke hvor god en sporing ville blitt.

Abstract

This thesis examins two algorithmic options for object detection of cars driving through Q-Free's tollstations. The algorithms presented in this thesis are YOLO and SSD which are both regarded as "State-of-the-art" object detection algorithms. Both algorithms are trained and evaluated using a dataset provided by Q-Free intailing videos captured by a widelenscamera positioned at their tollstations. For this thesis, only lightweight versions of the YOLO and SSD algorithms are reviewed to ensure that the final models can run on lightweight hardware and be used in future implementations to Q-Frees preexsisting tracking system.

The results discussed in this thesis contains both an evaluation of the trained models aswell as a visualization of the detections in images. Both algorithms achives great results when evaluating the predicitons made to detect cars driving through a tollstation. The results for both lies between 98% and 100% mAP and 98% F1 accuracy; however the visual video-results show's mistakes made by the model when predicting that would cause an unreliable tracking.

Figurer

Tabeller

Listings

Innholdsfortegnelse

1 Introduksjon og relevans

1.1 Bakgrunn

Prosessen med å detektere og spore en bil i dagens løsning innebærer å koble sammen data fra forskjellig tid og sted. Løsningen benytter et vidvinkelkamera for å koble sammen bak- og frontskiltene på kjøretøyene, samt en laser for klassifisering av type kjøretøy. Vidvinkelkameraet som benyttes har et bredt synsfelt, hvor kjøretøyene kan spores fra de ankommer sonen for lesing av bilskilt (ALPR-sonen) til de befinner seg under utstyrsportalen. Med det stadige økende bruksområde til maskinlæring, var det ønskelig for Q-Free å utforske mulighetene for å anvende deep learning i kombinasjon med vidvinkelkameraet for deteksjon. En nøyaktig deteksjon med lav feilrate er helt avgjørende for å kunne gjøre en god sporing.

1.2 Forskningsspørsmål

Målet for dette prosjektet er å finne gode alternativer til maskinlæringsalgoritmer for objektdeteksjon i sanntid, og sammenligne disse. Gjennom studiet ønsker vi å undersøke følgende spørsmål:

- Hvordan presterer maskinlæringsalgorimene for deteksjon av biler på Q-Free sitt datasett?
- Er det mulig å forbedre disse modellen for en bedre deteksjon?
- Hvilke scenarioer er det algoritmene gjør feil? Hva gjør bilene vanskelig å detekte?
- Hva er algoritmenes styrker og svakheter?

1.3 Rapportens struktur

Denne rapporten er bygd opp av følgende åtte kapitler:

Kapittel 1: Introduserer bakgrunnen og motivasjonen bak oppgaven, samt rapportens innhold.
Kapittel 2: Presenterer relevant teori for denne oppgaven. Det innebærer blant annet prinsipper innefor maskinlæring, samt feltene som omhandler objektdeteksjon og objektsporing.
Kapittel 3: Presenterer kjente objektdetekterings- og objektsporingsalgoritmer, samt tidligere arbeid som er gjennomført med disse.
Kapittel 4: Gir en beskrivelse av hvordan prosessen har foregått, inkludert en beskrivelse av benyttede biblioteker og verktøy.
Kapittel 5: Presenterer de oppnådde resultatene til de ulike modellene basert på valgene beskrevet i kapittel 4.
Kapittel 6: Tar for seg de oppnådde resultatene, og drøfter disse basert på valgte problemstilling.
Kapittel 7: Konkluderer oppgaven og gir en beskrivelse på hva som kan gjøres av

videre arbeid.

Kapittel 8: Gir en drøfting av etiske problemstillinger ved bruk og utvikling av maskinlæring.

1.4 Ordliste

Computer Vision datamaskinsyn. 2, 7

data augmentation dataøkning. 5, 20, 22

deep learning dyp læring. 4, 10, 25

feature extraction prosess for å hente ut trekk/kjennetegn fra data. 7, 11

features trekk/kjennetegn. 5–7, 10, 37

frame bilderamme. 10, 16, 19, 20, 23, 37, 38

frame rate bildefrekvens. 17

ground truth manuelt markert objekt. 8, 9, 11, 23, 31–34

labeling prosessen med å merke data med riktig klasse. 20

open source åpen kildekode. 7, 25

overfitting overtilpasning. 5, 21, 37

repository en lagringsplass for programvarepakker. 21, 23

supervised learning veiledet læring. 4

unsupervised learning ikke-veiledet læring. 5

1.5 Akronymer

AI Artificial Intelligence. 4

ALPR Automatic License Plate Recognition. 1

ANN Artificial Neural Network. 4–6

CNN Convolutional Neural Network. 6, 8, 16

CPU Central Processing Unit. 7, 25

CV Computer Vision. 7, 25, 40

DNN Deep Neural Network. 25

FPS Frames Per Second. 12, 14, 18

GPU Graphical Processing Unit. 7, 25, 26

IoU Intersection over Union. 9, 29

mAP Mean Average Precision. 9, 12, 14, 18, 20, 21, 27, 28, 36, 38, 39

MOT Multi-Object Tracking. 10, 18

OpenCV Open Source Computer Vision Library. 19, 25

SORT Simple Online and Real-time Tracking. 16

SSD Single Shot Detector. 14, 15, 18–20, 23, 25–29, 36, 39

YOLO You Only Look Once. 7, 11, 14, 18, 19, 21, 24, 26–29, 36, 38, 39

2 Teori

I dette kapittelet presenteres den teoretiske bakgrunnen for arbeidet som er gjennomført. Først gis en introduksjon av ulike maskinlæringsprinsipper og nevrale nettverk, etterfulgt av feltene innen maskinlæring som omhandler objektdeteksjon og objektsporing.

2.1 Maskinlæring

Kunstig intelligens er definert som et systems evne til å tolke ekstern data korrekt, lære fra denne dataen, og bruke det som er lært til å oppnå et spesifikt mål og/eller oppgave gjennom fleksibel tilpasning. [2]

Begrepet kunstig intelligens (eng. AI) ble først brukt i 1956 av Marvin Minsky og John McCarthy i deres forskningsprosjekt Dartmouth Summer Reasearch Project on Artifical Intelligence (DSRPAI). Prosjektet hadde som mål å skape et nytt forskningsområde med fokus på å lage maskiner med evnen til å simulere menneskelig intelligens. Dette ledet til utviklingen av programmer som *ELIZA*, et "natural language processing tool" med evnen til å ha en dialog med et menneske, og *The General Problem Solver* som hadde evnen til å løse enkle problemer som "Tårnene i Hanoi". [3]

Siden har kunstig intelligens hatt en stor utvikling, spesielt de siste årene med kunstig nevrale nettverk (ANN) og deep learning som går under begrepet maskinlæring. I 2015 gjorde kunstig intelligens et gjennombrud der programmet AlphaGo, med bruk av deep learning, slo verdensmesteren i brettspillet Go. Denne teknologien legger grunnlaget for kunstig intelligens slik den er i dag, og nevrale nettverk og deep learning brukes til blant annet bilde- og talegjenkjenning, klassifisering og deteksjon. [3]

Maskinlæring er en underkategori av kunstig intelligens (AI), og innebærer å lage applikasjoner som lærer fra data for deretter å forbedre sin nøyaktighet over tid uten at applikasjonen bli direkte programmert til å gjøre dette.

2.1.1 Supervised Learning

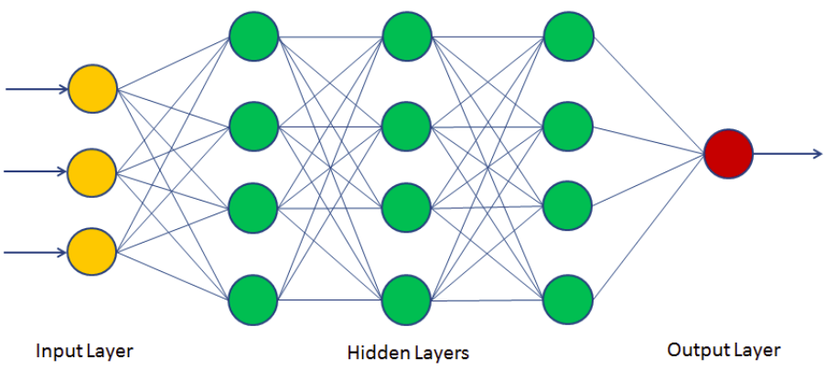
Ved *supervised learning* er det riktige svaret kjent på forhånd. Datasettet som brukes i treningen består av et *input*, samt den riktige *output*-verdien. Det er deretter opp til algoritmen å finne fram til den beste modellen som beskriver dette forholdet. Dette gjøres gjennom en treningsprosess der algoritmen inkrementelt endrer verdien til variablene i modellen basert på en tapsfunksjon som ser på forholdet mellom det reelle svaret som er gitt i datasettet og svaret som modellen gir. [4]

2.1.2 Unsupervised Learning

Ved *unsupervised learning* er ikke det riktige *outputet* kjent. I stedet er det algoritmen som forsøker å finne et mønster i *input*-dataen og grupperer dem basert på dette. Metoden fungere bra på å klassifisere kluster av data med attributter som ikke er lett for mennesker å klassifisere. [5]

2.2 Artificial Neural Network

Et *Artificial Neural Network (ANN)*, også kalt *Neural Network* (NN), er bygd opp av noder fordelt på tre typer lag: *input*-, *hidden*- og *output*-lag (Figur 2.1). Nodene i hvert av disse lagene er bygd opp av nevroner som inneholder forskjellige *features* fra *inputen*. Hvert nevron i et lag er koplet sammen med nevroner fra det foregående laget, og hver av disse koblingene har en tilhørende vekt. Når en prediksjon gjøres vil denne sammenlignes med sannheten ved bruk av en tapsfunksjon, som er et mål på presisjonen til nettverket. Det er ønskelig at tapsfunksjonen har en lavest mulig verdi. Etter at tapet er funnet, vil vektene korrigeres ved å økes eller reduseres basert på om den henholdsvis traff eller ikke ved å ta i bruk en optimaliserer. Denne prosessen kalles *gradient decent*. [6, 7]



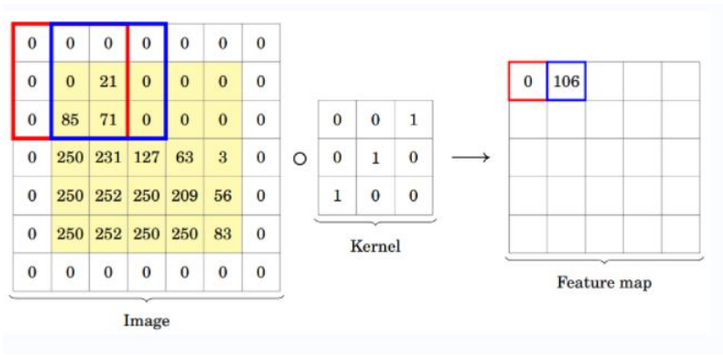
Figur 2.1: Oppbyggingen av et enkelt nevralt nettverk med noder fordelt på *input*-, *hidden*- og *output*-lagene, [6]

Et kjent problem for nevrale nettverk er at modellene tilsynelatende har god presisjon, men likevel presterer dårlig på testdataen. Dette fenomenet kalles *overfitting*. Fenomenet forekommer når modellen har er dårlig til å generalisere, det vil si at den har blitt for tilpasset treningsdataen og derfor ikke klarer å gjenkjenne *featuresene* i testdataen. For at en modell skal kunne generalisere bedre, trenger den å trenes på et variert og bredt datasett. I situasjoner hvor man har tilgang på begrenset og lite variert data, er det dermed vanlig å benytte *data augmentation*, en teknikk som øker datasettet ved å gjøre små endringer på den allerede eksisterende dataen. En annen strategi i forbindelse med *overfitting* er *Early Stopping*, og går ut på å stoppe treningen av en modell på et punkt hvor modellen ikke lenger forbedrer presisjonen. [8, 9]

2.3 Convolutional Neural Network

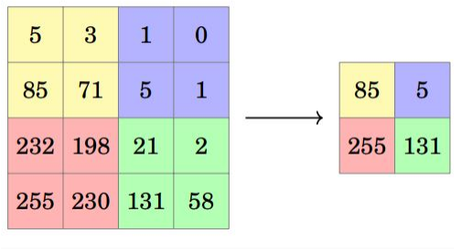
Convolutional Neural Network (CNN) er en variant av *ANN* som er egnet for bildeanalysering, da nettverket selv lærer seg å finne viktige *features* i data. Dette gjøres gjennom bruken av konvolusjons- og *max-pool-lag*. [10]

Konvolusjonslaget anvender en operasjon kalt konvolusjon for å trekke ut viktige *features* fra inputbildet. Dette foregår på den måten at ethvert element i en del av *input*-matrisen ganges med en verdi på tilsvarende posisjon i en $N \times N$-matrise kalt filter. Etter hver utregning blir verdiene summert og lagret i en *output*-matrise, og filteret flytter seg deretter et gitt antall *strides*. Dette er et mål på antall posisjoner/piksler filteret skal flyttes etter hver utregning. Denne prosessen repeteres til filteret har prosessert hele *input*-matrisen. Resultatet av denne operasjonen er en ny matrise med redusert dimensjon som inneholder de viktigste *featuresene* fra inputen. Denne matrisen kalles også *feature map*. [11]
Prosessen er visualisert i Figur 2.2. Her blir en 7×7-matrise redusert til en 5×5-matrise ved å benytte et filter med størrelse 3×3, samt en *stride* lik 1.



Figur 2.2: Prosessen konvolusjonslaget utfører for å trekke ut viktige *features* fra en *input*-matrise, [11]

Max-pool-lag forekommer som oftest mellom konvolusjonslagene, og sørger for å redusere størrelsen til dataen (matrisen) sendt som *input* ved å kun ta vare på elementene med høyest verdi. På denne måten vil de mindre relevante detaljene ignoreres slik at det kun er de mest fremtredende *featuresene* som representerer dataen. Antall parametre i nettverket vil dermed reduseres, noe som spesielt gjenspeiles i form av redusert utførelsestid. [11, 12]
I Figur 2.3 visualiseres prosessen som skjer når en 4×4-matrise sendes inn i et *max-pool-lag* med *stride* like 2 og et 2×2 filter.



Figur 2.3: *Max-pooling* på en $4x4$-matrise. Resultatet er en mindre matrise kun bestående av de høyeste verdiene, det vil si de viktigste trekkene fra *input*. [11]

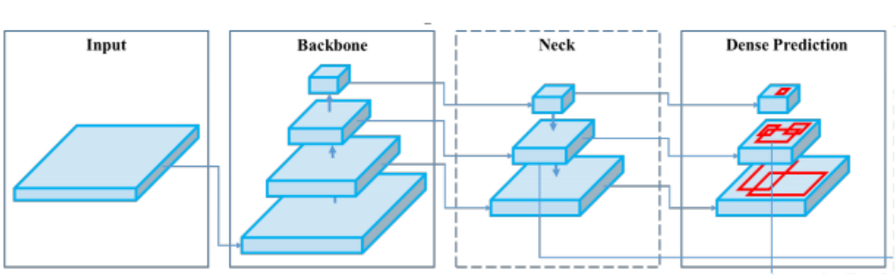
2.3.1 Darknet

Darknet er et *open source* rammeverk for nevrale nettverk, utviklet av Joseph Redmon. Rammeverket er kjent for å være raskt, og støtter både kjøring på CPU og GPU, noe som hovedsakelig kommer av at det er skrevet i C og CUDA. [13]
Ettersom Redmon også er en av utviklerne bak objektdetekteringsmodellen YOLO, er Darknet ofte brukt i forbindelse med trening av disse.

2.4 Objektdetektering

Objektdetektering har i løpet av de siste tiårene blitt en av de mest populære grenene innenfor Computer Vision (CV), mye på grunn av dets store bruksområde og teknologiske gjennombrudd i nyere tid [14, 15]. Formålet med objektdetektering er å lokalisere og klassifisere objekt(er) i et gitt bilde eller en video.

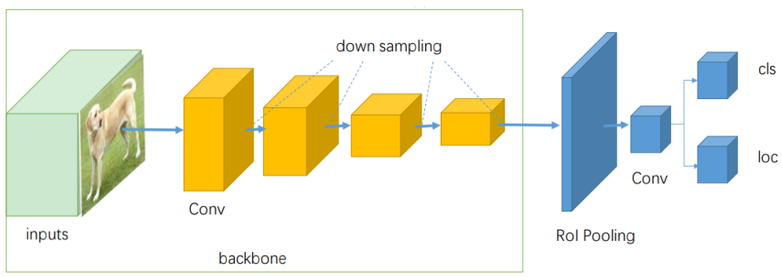
En objektdetekteringsmodell deles gjerne inn i tre deler: *backbone*, *neck* og *head* (Figur 2.4). *Backbone* er den delen av arkitekturen som tar seg av selve feature extraction. Til dette benyttes klassifiseringsmodeller som på forhånd er trent på store og varierte datasett. Features som blir funnet i backbone blir deretter koblet sammen i delen som kalles *neck*. Den siste delen av arkitekturen til modellen kalles *head*, *dense prediction* i Figur 2.4, og det er der selve predikeringene foregår. [16, 14]



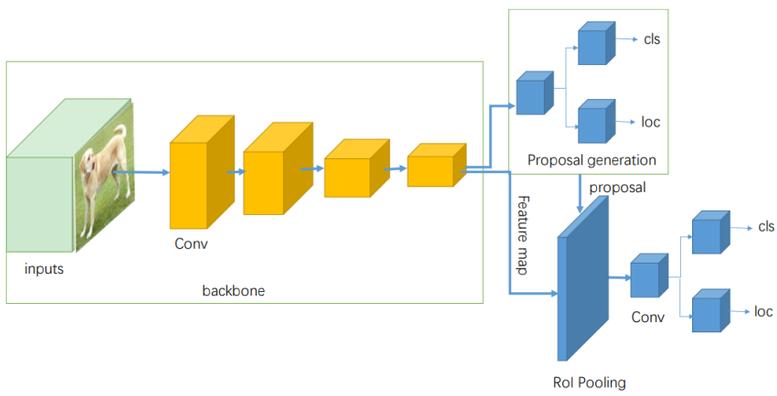
Figur 2.4: Oppbyggingen til en en-stegs objektdetekteringsmodell. [16, s. 2]

Hovedsakelig skilles det mellom to typer objektdetekteringsmodeller: en- og to-stegs-detektorer. Oppbyggingen av en- og to-stegs-detektorer kan sees i henholdsvis Figur 2.5(a) og 2.5(b). Fordelen til to-stegs-detektorene er at de har høy nøyaktighet når det kommer til lokalisering og klassifisering, og en-steg-detektorene er kjent for å være

effektive når det kommer til tidsbruk. I en to-stegs-detektor finner modellen såkalte *Regions of Interest* (RoI), områder i inputbildet som mest sannsynlig inneholder et objekt, som deretter sendes videre til et CNN som gjennomfører selve predikeringen av klasse og plassering gjennom bounding bokser. Denne regionbaserte CNN tilnærmingen kalles R-CNN. En-stegs-detektorer utelater steget med å finne de interessante områdene (RoI), og produserer i stedet prediksjonene direkte på input-bildet i form av *bounding* bokser. I tillegg til bounding boksene, oppgis en *confidence score* for hvilken klasse modellen tror objektet tilhører. Blant en-stegs-detektorene er SSD og YOLO de mest kjente [17]. [14]



(a) En-stegs-detektor



(b) To-stegs-detektor

Figur 2.5: Arkitekturen til en en-stegs-detektor og to-stegs-detektor. [14, s. 3]

2.4.1 Måling av presisjon

Som nevnt tidligere i delkapittel 2.4, predikeres forekomsten av et objekt gjennom bounding bokser. For hver prediksjon inkluderes informasjon om plassering, klasse, samt et mål kalt *confidence score* på hvor sikker denne prediksjonen er. Evaluering av en objektdekekteringsmodell er basert på hvor godt de predikerte *bounding* boksene samsvarer med ground truth.

Precision og Recall

Precision og *Recall* er to sentrale verdier for evaluering av en objektdetekteringsmodell. Disse er henholdsvis gitt som:

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

hvor TP = *True Positive*, FN = *False Negative*, FP = *False Positive*. TP betyr at modellen har klassifisert et objekt riktig. FN betyr at modellen ikke har klassifisert et objekt som er der. FP betyr at modellen klassifiserer et objekt som ikke er der. [18]

Intersection over Union

Intersection over Union (IoU) defineres som

$$IoU = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$

og er et mål på hvor stort det overlappende området er mellom den predikerte bounding boksen og ground truth. En deteksjon betegnes som god jo nærmere IoU er lik 1, da avviket mellom prediksjon og sannhet er mindre i disse tilfellene. [19]

Under deteksjon, og trening, kan det settes en IoU-grense for hva som skal bestemme om en prediksjon er TP, FN eller FP. Eksempelvis kan denne verdien være satt til 0.5. Dette vil si at om det overlappende området mellom bounding boks og ground truth har $IoU < 0.5$ eller ved duplisert bounding boks vil prediksjonen bli en False Positive. Om $IoU > 0.5$ vil den bli True Positive. Ved en prediksjon $IoU > 0.5$, men feil klassifisering eller ingen deteksjon hvor det skal være er det en False Negative. [18]

F1-score

F1-score er et annet mål på nøyaktigheten til en modell som kombinerer *Precision* og *Recall* i utregningen:

$$F1 = 2 \cdot \frac{Precision \times Recall}{Precision + Recall}$$

I likhet med IoU vil en F1-score nærmere 1 være en indikasjon på at modellens prestasjon er god [20]. Det vil si at man har et lavt antall FN og FP, noe som er ønskelig da majoriteten av deteksjonene er sanne (TP).

Mean Average Precision

Mean Average Precision (mAP) er en populær metode for å måle nøyaktigheten til deteksjonsmodeller ved å bruke Precision, Recall og IoU i utregningen. mAP er regnet ut fra arealet under Precision-Recall(PR)-kurven, noe som kan gjøres for ulike IoU-grenser. Eksempler på vanlige IoU-grenser er 0.50 og 0.75, og oppgis som henholdsvis mAP@50 og mAP@75. En rangering av konfidensnivået til prediksjonene i avtagende rekkefølge gir en funksjon som PR-kurven plottes ut fra. PR-kurven vil få en nedgående bane, fordi for hver påfølgende rank vil konfidensnivået være lavere. Dette resulterer i at sannsynligheten for korrekt deteksjon er lavere. *Recall* øker for flere prediksjoner, da den aldri reduseres, samt øker for alle TP. [21]

2.5 Objektsporing

Et annet viktig område innenfor datamaskinsyn er objektsporing. Prosessen innebærer å spore objekter gjennom flere *frames* i en video, hvor objektene på forhånd er detektert ved hjelp av en objektdetekteringsalgoritme. Denne metoden omtales ofte som *tracking-by-detection*. For å kunne identifisere det samme objektet i en påfølgende *frame* får hvert objekt tildelt en unik ID. Populære bruksområder for objektsporing er overvåkning, analysering av trafikk og selvkjørende biler. [22, 23]

Man skiller gjerne mellom to former for objektsporing: *online* og *offline* sporing. *Online* objektsporing foregår på sanntidsvideo. Det betyr at denne metoden kun har tidligere informasjon i form av *frames* å gå ut ifra. Ettersom fremtidige *frames* ikke er kjent, i tillegg til at algoritmen i større grad må jobbe raskere, ses *online* sporing som mer krevende prosess sammenlignet med *offline* sporing. *Offline* sporing forenkler prosessen ved at sporingen foregår på en allerede innspilt video, noe som betyr at både tidligere og fremtidige *frames* er kjent på forhånd. [24]

Objektsporing kan ses på som en mer kompleks prosess sammenlignet med objektdetektering, da det medfører noen problemstillinger som må tas hensyn til. Eksempler på slike problemstillinger er hvordan objekter fra en *frame* skal kobles sammen til den påfølgende *frame*, samt hvordan situasjoner hvor to objekter krysser/overlapper hverandre skal løses. Tilsvarende utfordringer må spesielt tas hensyn til i Multi-Object Tracking (MOT). [25]

I MOT foregår det sporing på flere objekter samtidig. Problemstillinger som presenteres i MOT er blant annet hvordan lokalisering og identifisering av objektene skal håndteres i situasjoner hvor objektene er vanskelig å skille eller overlapper hverandre. Framgangsmåten i MOT skilles mellom to ulike grener: *Detection-Based Tracking* (DBT) og *Detection-Free Tracking* (DFT). DBT er en *tracking-by-detection*-metode som, beskrevet tidligere i delkapitellet, detekterer objektene ved hjelp av en objektdetekteringsmodell før de kan spores. Dette innebærer at objektdetekteringen legger en stor del av grunnlaget for prestasjonen til sporingen. Bruken av *deep learning* i detekteringsfasen har i løpet av de siste årene blitt sett på som en metode for å øke prestasjonen på sporingen betraktelig, mye på grunn av nettverkenes evne til å finne og trekke ut mer omfattende *features* fra *input* [26]. I den andre fremgangsmåten, DFT, blir ikke en objektdetekteringsmodell involvert, noe som betyr at objektene man ønsker å spore i påfølgende *frames* må manuelt markeres i den første *frame*. [27, 28]

3 Tidligere arbeid

I dette kapittelet presenteres kjente algoritmer for objektdeteksjon og objektsporing, samt tidligere relevant arbeid som er gjort i forbindelse med disse.

3.1 You Only Look Once

You Only Look Once (YOLO) er en en-stegs-objektdetekteringsalgoritme som er kjent for dens enkle oppbygging og raske detektering Ettersom YOLO tilhører kategorien en-stegs-detektor behøver den, som nevnt i delkapittel 2.4, kun å se på et bilde én gang for å kunne predikere og navngi hvilke objekter som befinner seg på dette bildet. Dette foregår ved at inputbildet deles inn i et à $S \times S$ celler, hvor det i hver celle blir satt opp et gitt antall *anchor* bokser. For hver av disse boksene beregnes en *confidence score*, gitt som

$$\text{Confidence score} = P(Object) \cdot IoU$$

hvor $P(Object)$ er et mål på hvor sikker den er på at boksen inneholder et objekt, og IoU (delkapittel 2.4.1) måler nøyaktigheten på den predikerte boksen ved å se på forholdet mellom denne og den faktiske boksen (*ground truth*). *Confidence scoren* til hver genererte boks blir deretter målt opp mot et gitt *confidence threshold*. Dersom *confidence score* er høyere enn *confidence threshold*, beholdes boksen. På denne måten er det kun boksen med høyest IoU med *ground truth* som til slutt representerer objektet. [29]

YOLO utfører detekteringsprosessen på et gitt antall lag i nettverket med forskjellige skaleringer, basert på hvilken YOLO-versjon som benyttes. Dette foregår ved at inputbildet blir delt inn i et rutenett à $S \times S$ hvor S er definert som størrelse på inputbildet dividert med en *stride*. For hver av de ulike rutenett-størrelsene benyttes også passende *anchor/bounding* bokser. Dette betyr at små, medium og store *anchor* bokser benyttes når S henholdsvis har en høyere, medium, respektiv lavere verdi. [30]

3.1.1 YOLOv4

Den fjerde versjonen av objektdetekteringsalgoritmen YOLO, YOLOv4, ble introdusert av Bochkovskiy et al. (2020). En av modifikasjonene i YOLOv4 var blant annet å utvide Darknet53 i *backbone* med et CSPNet, kalt CSPDarknet53. I *neck*-delen av nettverket ble teknikker som SPP (*Spatial Pyramid Pooling*) og PANet (*Path Aggregation Network*) anvendt for å forbedre *feature extraction*. Til tross for endringene gjort i *backbone* og *neck*, benytter modellen seg av konsepter fra den foregående versjonen (YOLOv3) i *head*. Anvendelsen av de nye teknikkene førte til at YOLOv4 ble ansett som den beste tilgjengelige objektdetektoren. [16]

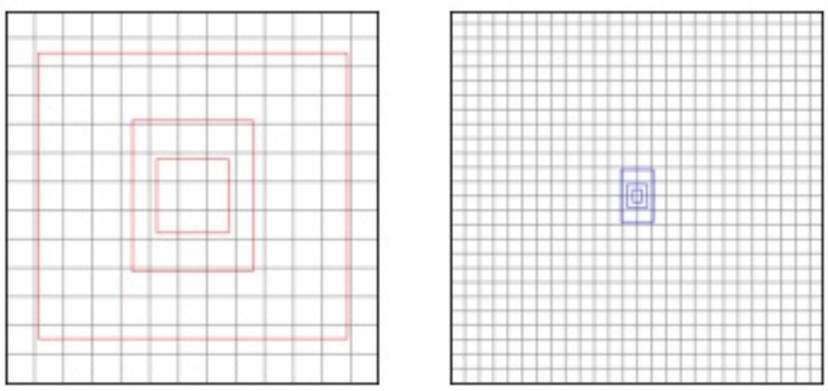
3.1.2 YOLOv4-Tiny

For å tilpasse YOLOv4 til mobile og innebygde enheter, ble YOLOv4-Tiny utarbeidet. Forenkling av nettverkstrukturen og redusering av antall parametre er justeringer som ble gjort for å gjøre denne versjonen mer kompatibel for enklere systemer. Eksempler på områder lettvektsmodeller, som YOLOv4-Tiny, er tatt i bruk er blant annet ved detektering av kjøretøy. Gjennom å bruke CSDarknet53-tiny som *backbone* i stedet for CSPDarknet53, som YOLOv4 er basert på, ble modellens kompleksitet redusert. Se Figur 3.2 for nettverkstrukturen til YOLOv4-Tiny. Resultatet av endringene gjort av Jiang et al. (2020) [31] synliggjøres særlig gjennom antall bilder som kan detekteres per sekund (FPS) (Tabell 3.1). Prestasjonen til de ulike modellene i denne tabellen er målt ut fra oppnådd *FPS* og mAP på datasettet Microsoft Common Object in Context (MS COCO). Datasettet består av over 122000 trenings- og testbilder, fordelt ut over 80 klasser. Som følge av de nevnte tilpasningene presterte YOLOv4-Tiny bedre på *FPS* sammenlignet med YOLOv4, men dette gikk samtidig på bekostning av modellens gjennomsnittlige presisjon (mAP). [31]

Metode	FPS	mAP(%)
YOLOv3	49	52.5
YOLOv4	41	64.9
YOLOv3-tiny	277	30.5
YOLOv4-tiny	270	38.1

Tabell 3.1: Sammenligning av FPS og mAP for ulike YOLO-versjoner. [31, s.7]

YOLOv4-Tiny utfører detektering på to *scales*, med *strides* på 32 og 16 [31]. Som beskrevet i delkapittel 3.1 vil dermed et inputbilde med størrelse 416×416 gi to deteksjonslag med størrelse på henholdsvis 13×13 og 26×26. I YOLOv4-Tiny benyttes totalt seks *anchor* bokser, tre i hver av de nevnte deteksjonslagene [13]. I Figur 3.1 er de to deteksjonlagene tegnet med deres respektive *anchor* bokser. Sett i sammenheng med delkapittel 3.1, vil YOLOv4-Tiny i hovedsak kunne detektere store (røde *anchor* bokser) og medium store (blå *anchor* bokser) objekter.



Figur 3.1: Visualisering av deteksjonslagene på 13×13 celler og 26×26 celler med tilhørende *anchor* bokser

Layer	Type	Filters	Size/Stride	Input	Output
0	Convolutional	32	$3 \times 3/2$	$416 \times 416 \times 3$	$208 \times 208 \times 32$
1	Convolutional	64	$3 \times 3/2$	$208 \times 208 \times 32$	$104 \times 104 \times 64$
2	Convolutional	64	$3 \times 3/1$	$104 \times 104 \times 64$	$104 \times 104 \times 64$
3	Route 2				
4	Convolutional	32	$3 \times 3/1$	$104 \times 104 \times 32$	$104 \times 104 \times 32$
5	Convolutional	32	$3 \times 3/1$	$104 \times 104 \times 32$	$104 \times 104 \times 32$
6	Route 5 4				
7	Convolutional	64	$1 \times 1/1$	$104 \times 104 \times 64$	$104 \times 104 \times 64$
8	Route 2 7				
9	Maxpool		$2 \times 2/2$	$104 \times 104 \times 128$	$52 \times 52 \times 128$
10	Convolutional	128	$3 \times 3/1$	$52 \times 52 \times 128$	$52 \times 52 \times 128$
11	Route 10				
12	Convolutional	64	$3 \times 3/1$	$52 \times 52 \times 64$	$52 \times 52 \times 64$
13	Convolutional	64	$3 \times 3/1$	$52 \times 52 \times 64$	$52 \times 52 \times 64$
14	Route 13 12				
15	Convolutional	128	$1 \times 1/1$	$52 \times 52 \times 128$	$52 \times 52 \times 128$
16	Route 10 15				
17	Maxpool		$2 \times 2/2$	$52 \times 52 \times 256$	$26 \times 26 \times 256$
18	Convolutional	256	$3 \times 3/1$	$26 \times 26 \times 256$	$26 \times 26 \times 256$
19	Route 18				
20	Convolutional	128	$3 \times 3/1$	$26 \times 26 \times 128$	$26 \times 26 \times 128$
21	Convolutional	128	$3 \times 3/1$	$26 \times 26 \times 128$	$26 \times 26 \times 128$
22	Route 21 20				
23	Convolutional	256	$1 \times 1/1$	$26 \times 26 \times 256$	$26 \times 26 \times 256$
24	Route 18 23				
25	Maxpool		$2 \times 2/2$	$26 \times 26 \times 512$	$13 \times 13 \times 512$
26	Convolutional	512	$3 \times 3/1$	$13 \times 13 \times 512$	$13 \times 13 \times 512$
27	Convolutional	256	$1 \times 1/1$	$13 \times 13 \times 512$	$13 \times 13 \times 256$
28	Convolutional	512	$3 \times 3/1$	$13 \times 13 \times 256$	$13 \times 13 \times 512$
29	Convolutional	21	$1 \times 1/1$	$13 \times 13 \times 512$	$13 \times 13 \times 21$
30	YOLO				
31	Route 27				
32	Convolutional	128	$1 \times 1/1$	$13 \times 13 \times 256$	$13 \times 13 \times 128$
33	Upsample		2x	$13 \times 13 \times 128$	$26 \times 26 \times 128$
34	Route 33 23				
35	Convolutional	256	$3 \times 3/1$	$26 \times 26 \times 384$	$26 \times 26 \times 256$
36	Convolutional	21	$1 \times 1/1$	$26 \times 26 \times 256$	$26 \times 26 \times 21$
37	YOLO				

Figur 3.2: Nettverkstrukturen til YOLOv4-Tiny. Deteksjonslagene er i denne figuren kalt YOLO, og foregår på lag 30 og 37. Første YOLO-lag har inputstørrelse 13×13, og andre YOLO-lag har inputstørrelse 26×26. [32]

3.1.3 Sanntidsdeteksjon med YOLO i bomstasjon

Et av områdene som har sett på muligheten til å anvende maskinlæring i form av objektdetektering er automatiske bomstasjoner. Et studie av D. Chattopadhyay et al. (2020) så på bruken av maskinlæring for sanntidsdeteksjon av kjøretøy i automatiske bomstasjoner, og ønsket å undersøke om dette kunne erstatte eksisterende løsning. Studiet pekte på at infrastrukturen til den eksisterende løsningen både var kompleks og dyr, og mente at dette kunne reduseres gjennom å anvende et *overhead*-kamera. For selve deteksjonen ble objektdetekteringsmodellen YOLOv3 og YOLOv3-Tiny benyt-

tet. De høyeste resultatene for mAP som ble oppnådd med den foreslåtte løsningen med *overhead*-kamera og objektdetekteringsmodellen, var på 55.6% og 45.5% for henholdsvis YOLOv3 og YOLOv3-Tiny. I tillegg til vurdering av mAP-verdi, sammenlignet studiet deteksjonstiden til de to modellene. Til sammenligning med YOLOv3 sin deteksjonstid på 104.4ms, hadde YOLOv3-Tiny en deteksjonstid på kun 27.6ms. Basert på de overbevisende resultatene, konkluderte studiet med at den eksisterende, komplekse infrastrukturen kunne erstattes av den foreslåtte løsningen. Det ble i tillegg påpekt at YOLOv3-Tiny var å anbefale for dette spesifikke problemet, da den hadde en betydelig lave deteksjonstid sammenlignet med YOLOv3. [33]

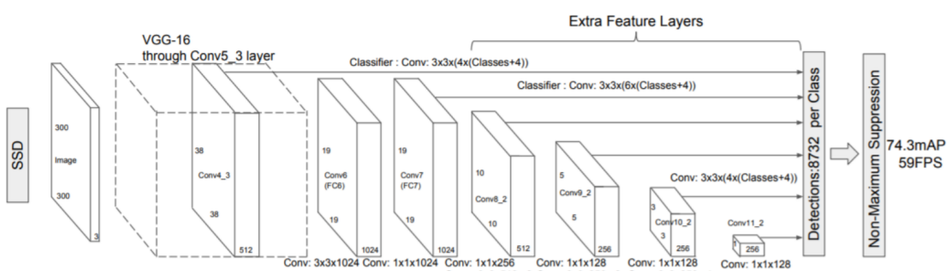
3.2 Single Shot Detector

Single Shot Detector (SSD) er en lettvektsalgoritme med god presisjon og effektivitet [34]. Da SSD først ble presentert i 2016 viste resultatene en betydelig forbedring fra tidligere modeller som Faster R-CNN og den daværende YOLO-versjonen. På VOC2007 testen viste SSD stor forbedring for både deteksjons hastighet og deteksjons presisjon med en hastighet på 59 FPS og mAP på 74,3%. [35]

I dag er SSD fremdeles en av de mest effektive objektdeteksjons-algoritmene, og er kjent for å ha høy presisjon i forhold til hastighet. [36]

3.2.1 SSD Oppbygging

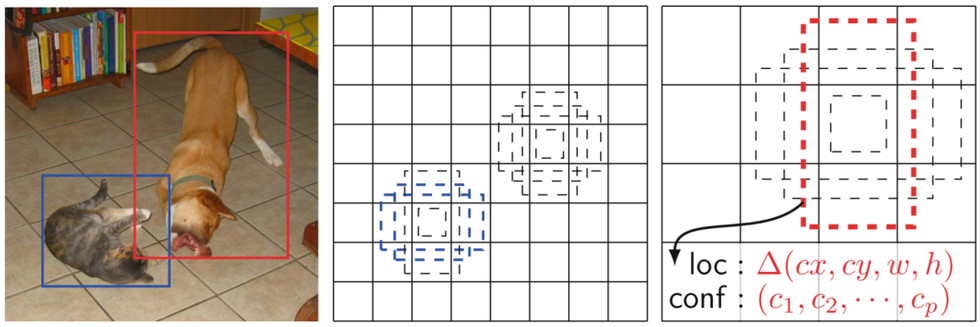
SSD-strukturen består overordnet av to deler, en forhåndstrent klassifiseringsmodell som legger grunnlaget for strukturen (*backbone*) og ett SSD hode som produserer deteksjonene (*head*). Når SSD først ble presentert ble VGG-16 nettverket brukt som *backbone*, men det har siden blitt brukt flere alternativer som ResNet og MobileNet. Strukturen til SSD-modellen med VGG16 som *backbone* og *input*-størrelse på 300×300 kan ses i Figur 3.3. [35] Se Figur 3.3.



Figur 3.3: Arkitekturen til SSD

De første lagene i *backbone* av modellen er konvolusjonslag som utfører *feature map* ekstraksjon. *Feature maps* er definert som resultatet av å anvende lagene med sine respektive vekter på inputbildet [37]. Disse lagene vil hente ut sentrale trekk fra inputbildet. De nederste lagene vil typisk identifisere ting som kanter og vertikale/horisontale linjer, mens øvre lag gjenkjenner mer spesifikke trekk som for eksempel et bilhjul.

SSD anvender *Multi-scale feature maps* for deteksjonen. Disse lagene i modellen utfører objektdeteksjon ved å sende resultatet fra *backbone* gjennom nye konvolusjonslag som gradvis avtar i størrelse. Hvert lag kan produsere en prediksjon som tillater deteksjon av objekter i ulik størrelse. For å produsere en prediksjon bruker SSD et lite sett med standardbokser, også kalt *default* bokser, med ulik skalering. Disse evalueres for hver lokasjon i de ulike *feature mapene* med varierende størrelse. Et overordnet eksempel på dette er illustrert i Figur 3.4



(a) Image with GT boxes (b) 8×8 feature map (c) 4×4 feature map

Figur 3.4: SSD deteksjon med med *Feature maps* og *Default* bokser

3.2.2 Sanntidsdeteksjon med SSD for Sikkerhetssytemer

I 2021 presenterte Saji og Sobhana en artikkel kalt "Real Time Object Detection Using SSD for Bank Security". I en artikkel av Saji og Sobhana (2021) presenteres en foreslått metode for et sikkerhetssytem for bank og lignende industrier. Denne metoden bruker objekt deteksjon med SSD for å detekte objekter fra et webkamra for å identifisere og spore elementer som utgjør en trussel. SSD-modellen som er presentert bruker MobileNet v2 som *backbone* og en *lite* versjon av SSD. Modellen er testet på COCO datasettet. Resultatene som er presentert viser at deres modell presterer betydelig bedre enn de andre modellene i testen, Faster-rcnn-inception-v2, Faster-rcnn-resnet50 og Rfcn-resnet101. Målt F1-verdi er på 93% for den foreslåtte modellen SSDlite-MobileNet-v2. [36]

3.2.3 Multi-Block SSD for overvåking

I en artikkel av Li et al.(2019) foreslås en *Multi-block* SSD-metode for å forbedre SSD-modellens evne til å detekte mindre objekter i bilder. Dette er kjent som en av svakhetene med SSD. Forbedringen som er foreslått innebærer å dele opp det originale bildet i flere overlappende del bilder som så individuelt blir sendt gjennom SSD nettverket. Testene som blir presentert sammenligner den foreslåtte modellen med standard SSD [35] og en forbedret SSD [38]. Resultatene viser en betydelig forbedring for prediksjon på deres datasett med en gjennomsnitt F1-verdi på 96.6% for deres *Multi-block* SSD mens standard SSD og forbedret SSD har en gjennomsnittlig F1-verdi på henholdsvis 87.4% og 61.5%. [34]

3.2.4 MobileNet

MobileNet er en effektiv nettverksarkitektur som ble først presentert i 2017 som et alternativ for små og effektive nevrale nettverk for bruk i applikasjoner der hastighet er viktig. Dette gjelder for eksempel tilfeller som krever sanntidsdeteksjon og/eller kjøring på lettvekt maksiner. [39]

3.3 DeepSORT

DeepSORT ble presentert av Wojke et al. (2017) som en forbedret versjon av sporingsalgoritmen SORT (Simple Online and Real-time Tracking) [23]. DeepSORT betegnes som *tracking-by-detection* metode. En deteksjonalgoritme blir kjørt på hver *frame* av en video, og bruker deteksjonene som er blitt gjort for å spore objekter fra en *frame* til den neste. Denne typen sporing bruker en åtte-dimensjons tilstandsvektor $(x, y, \gamma, h, {}' x, {}' y, {}' \gamma, {}' h)$ som representerer senter-posisjonen til en *bounding* boks (x, y), høyden (h), størrelsesforhold (γ), og til slutt deres hastighet i bildekoordinater. Banen oppdateres ved bruk av Kalman Filter. Kalman Filter er en algoritme som først ble presentert på slutten av 1950-tallet av Rudolf E. Kalman. Algoritmen baserer seg på å få *input* fra ulike målinger for så å bruke disse til å estimerer tilstanden X. Kalman Filter blir brukt i situasjoner der variablene man ønsker å beregne ikke kan bli målt direkte og når det er ønskelig å ta i bruk flere målinger med støy for å estimere tilstanden [40]. Ved å se på den tidligere posisjonen til et objekt, gjennom å anvende lineær observasjon og konstant hastighet, kan Kalman Filter estimere posisjonen til et (sporet) objekt i en gitt *frame*. Dette kombineres med Hungarian-algoritmen [41], som kobler sammen et objekt fra den forrige og nåværende *frame*, noe som ofte kan ses ved at et objekt gjerne spores med en tilhørende ID. [23, 42]

Hver deteksjon i en *frame* blir tildelt en *track*. Her lagres nødvendig informasjon. Når en deteksjon først ankommer *frame* har DeepSORT et minimum antall deteksjoner som må til før en *track* blir tilegnet en deteksjon. Denne mekanismen er implementert for å forhindre duplikat sporing. Det er også lagret informasjon om sist en *track* ble detektert. Dette *tracket* vil bli slettet etter denne grensen er nådd. Dette er for å slette *tracks* som har forlatt *frame*. [43]

En av svakhetene til forgjengeren, SORT, var at det til tross for god presisjon og nøyaktighet på sporingen, ofte forekom såkalte identitetsvitjer. Dette er situasjoner hvor et objekt går tapt og detekteres som et nytt objekt under sporingen. Ved å benytte et CNN i kombinasjon med Kalman Filter og Hungarian-algoritmen, har DeepSORT klart å redusere antall identitetsvitsjer, samt blitt i stand til å spore objekter i løpet av en lengre periode. Beregninger gjort av Wojke et al. (2017) viste at DeepSORT reduserte antall identitetsvitsjer med 45% sammenlignet med SORT. [23]

3.3.1 Objektsporing og objektdeteksjon

Sporing er sterkt tilknyttet deteksjon. Det vil si at uansett hvor kompleks sporingalgoritmen er, vil den gi et dårlig resultat om deteksjonen er dårlig. Det er de siste årene blitt store forbedringer i deteksjonsalgoritmer. Denne trenden ser ikke til å snu, og ved videre utvikling er det en tydelig tendens at sporing vil bli svært nøyaktig og effektiv. Bedre deteksjon vil også føre til at sporingsalgoritmene i seg selv ikke trenger å være i veldig tunge prosesser så lenge deteksjon er bra og videoen som blir sendt inn har høy *frame rate*.

Mandal og Adu-Gyamfi (2020) la i 2020 fram en sammenligning av en rekke kombinasjoner av objektdetekteringsmodeller og sporingalgoritmer for kjøretøy. Modellene ble testet på videoklipp, fra forskjellige tider på døgnet, på til sammen ni timer. Prestasjonen til de ulike kombinasjonene ble basert på hvor mange prosent av kjøretøyene de klarte å telle sammenlignet med det faktiske antallet, og ble gjort for to forskjellige kameravinkler. Resultatet av analysen viste at kombinasjonene hvor DeepSORT ble benyttet var de som presterte best på begge kameravinklene. [44]

4 Metode og teknologi

I dette kapittelet begrunnes valgene av metode og teknologi. Metodekapittelet tar for seg prosessen fra utarbeiding av datasett, valg og trening av modellene, og til slutt utføring av selve deteksjonen. Videre presenteres ulike biblioteker og relevante verktøy som er benyttet i implementerings- og treningsfasen av maskinlæringsmodellen.

4.1 Metode

Fokuset under studiet var å finne tilgjengelige objektdetekteringsmodeller med god effektivitet og nøyaktighet, samt støtte for en lettvektsversjon. Det eksisterer mange forskjellige modeller for objektdeteksjon, som stadig kommer med nyere, forbedrede versjoner (delkapittel 2.4). En-stegs-detektorer (delkapittel 2.4), er modeller som ved kun et steg kan detektere flere objekter i et bilde. Disse metodene er kjent for å være effektive og nøyaktige. To kjente, og mye brukte en-stegs-detektorer er SSD og YOLO. Som nevnt i delkapittel 3.1 og 3.2 er modellenes styrker henholdvis effektivtet og nøyaktighet. I tillegg har begge modellene en lettvektsversjon. Ut fra nevnte kvaliteter ble disse metodene valgt for implementasjon og sammenligning i dette studiet.

Det ble i tillegg gjort undersøkelser på objektsporingsalgoritmer, hvor det var ønskelig med en stabil og konkurransedyktig løsning på MOT-problemer. DeepSORT har som nevnt i delkapittel 3.3 hatt gode resultater for sporing av kjøretøy, så det ble derfor ønsket å ta i bruk denne for å løse sporingsproblemet. Sporingen av biler ble dog valgt som sekundærprioritet til selve deteksjonen av bilene. Dette valget ble tatt med den begrunnelse at god deteksjon er fundamentalt for god sporing.

4.1.1 Valg av SSD-modell

SSD algoritmen er kjent for god deteksjon og hastighet, og har vist gode resultater ved bruk i lignende problemstillinger. Som presentert i delkapittel 3.2.2, viste en SSD MobileNet-modell en nøyaktighet på 93% på COCO datasettet. Denne modellen ble også sagt å ha en god ytelse på systemer med mindre prosessor kraft og en rask predik-sjons hastighet. SSD er ofte brukt med MobileNet som *backbone*. Dette gjelder typisk for situasjoner der modellen skal kjøre på mindre maskiner som for eksempel en mobil eller brukes til sanntids deteksjon der effektivitet er viktig.

4.1.2 Valg av YOLO-modell

YOLOv4 og YOLOv4-Tiny ble funnet til å være en klar forbedring fra YOLOv3 og YOLOv3-Tiny, basert på oppnådd mAP og FPS (Tabell 3.1). I tillegg til å ha en høyere mAP enn YOLOv3-Tiny, oppnådde YOLOv4-Tiny et høyere antall FPS enn forgjengeren. Tidligere arbeid på lignenede område, som beskrevet i kapittel 2.4, viste at YOLOv3-Tiny målte seg med den eksisterende løsningen. Det ble derfor sett på som interessant å undersøke hvordan YOLOv4-Tiny ville prestere på et lignende scenario.

4.1.3 Datasett

Datasettet som er brukt i prosjektet er generert fra videofiler gitt av Q-Free. Videoene er fra Q-Free sine bomstasjoner, og består av video tatt fra begge sider av bomstasjonen med variasjon i værforhold og trafikkmengde. Variert data i datasettet er sentralt for å trene en god og robust modell (delkapittel 2.2). For generering av *frames* fra videofilene ble videoprosseseringsmetoder fra OpenCV-biblioteket benyttet. Hvert bilde er deretter manuelt *lablet*/klassifisert i programmet LabelImage [45], der hver instans av en bil er markert. I tillegg til å effektivisere prosessen med generering av datasett, tilbyr programmet støtte for annoteringen på både PASCAL- og YOLO-formatet som brukes for henholdsvis SSD og YOLO.

Totalt er det tatt utgangspunkt i 24 videofiler i svart-hvitt for generering av *frames*. Disse ble deretter fordelt på trenings-, valideringsdata og testdata. Det har blitt trent modeller på to forskjellige datasett, ett med og ett uten tomme veier, som en undersøkelse på om dette hadde en innvirkning på modellenes nøyaktighet. Fordelingen av dataen i de to datasettene vises i Tabell 4.1. For utarbeiding av valideringsdataen ble det tatt utgangspunkt i bilder hvor man kunne følge en bil/flere biler i det den/de passerer kjører inn/ut fra bomstasjonen. Dette ble hovedsakelig gjort som et forsøk på å unngå at data fra den samme videofilen ble benyttet både i trening og validering.

	Trening	Validering	Testing	Totalt
Med tomme veier	2757	799	176	3732
Uten tomme veier	1714	719	176	2609

Tabell 4.1: Fordeling av data på datasett med og uten tomme veier

4.1.3.1 SSD-format

For SSD var det nødvendig å konvertere *labelformatet* fra YOLO til PASCAL-formatet. På dette formatet er *label*-informasjonen gitt i XML-filer, med én fil for hvert *lablet* bilde. Filen gir informasjon om bildets dimensjoner og lokasjon, samt koordinater og klassifisering av hvert *lablet* objekt som finnes i bildet. For konverteringen ble det brukt et Python-*script* skrevet av Chanjoo Lee [46].
For å kunne ta i bruk TensorFlow sitt *Object Detection API* var det nødvendig å lagre trening og valideringsdataen som *TensorFlow Records* (TFRecords). Dette er TensorFlow sitt eget format som er laget for å enklere kunne håndere data under trening [47]. XML-filene ble først lagret i en CSV-fil ved bruk av følgende kode "xml_to_csv.py" skrevet av Khush Patel [48]. Trenings- og valideringsdata ble lagret i hver sin CSV-fil for å forsikre at den samme bildedataen ble brukt til trening og validering for både YOLO og SSD modellen. Fra CSV-filene ble det laget én TFRecord for treningsdataen og én for test/valideringsdataen. For generering av TFRecords ble følgene kode brukt "generate_tfrecord.py" som er en modifisert kode fra *TF Object Detection API* modifisert av Dat Tran [49].

4.1.3.2 YOLO-format

For YOLO ble YOLO-formatet bruk under *labeling*, da det var dette formatet Darknet krevde. For hvert *lablet* bilde som opprettes en *.txt* fil som inneholder klassen etterfulgt av x- og y-koordinatene til midten, høyden og bredden til objektet, eventuelt objektene, som befinner seg i gitt *frame*:

<klasse> <x> <y> <bredde> <hoyde>

Et eksempel på en tekstfil med YOLO-format i dette datasettet er gitt i Listing 4.1. Her indikerer `<klasse>` lik 0 at objektet er en bil.

```
0 0.157407 0.573958 0.111111 0.102083
```

Listing 4.1: Eksempel på fil med YOLO-format

4.1.4 Trening med SSD

Trening av SSD-modellene ble gjort ved bruk av *TensorFlow Object Detection API*. APIet tilbyr et rammeverk basert på TensorFlow som skal sørge for enkel trening og eksportering av modeller [50].

For å spare tid og ressurser baserer treningen seg på forhåndstrente modeller hentet fra TensorFlow sin modell Zoo [51]. Det ble testet med flere ulike modeller som baserer seg på MobileNet nettverket. MobileNet ble valgt fordi det er kjent for høy hastighet og god presisjon, som er ønskelig i dette prosjektet der tenkt sluttprodukt skal kunne kjøres på Coral USB Accelerator [1] med sanntidsdeteksjon og sporing. De forhåndstrente SSD-modellene som tar i bruk MobileNet viste også god resultater på hastighet og mAP. Alle de forhåndstrente modellene som er testet, er trent på COCO-datasettet som inneholder 91 ulike objektkategorier med 328 000 bilder med 2.5 millioner lablet objekter [52]. De forhåndstrente modellene som ble testet er følgende:

- ssd_mobilenet_v1_fpn_640x640, mAP COCO: 29.1%
- ssd_mobilenet_v2_320x320, mAP COCO: 20.2%

Fra TF Object Detection APIet ble det tatt i bruk `model\_main\_tf2.py` for å kjøre en trening. Dette scriptet bruker en *pipeline.config* fil for å konfigurere treningen. Denne filen inneholder blant annet informasjon om antall klasser, *input* størrelse, antall iterasjoner og filplassering til trening og test dataen. Her er det også definert eventuell *data augmentation*.

SSD_mobilenet_v1, trening uten tomme veier

- Bilde *Input* Størrelse: 640x640
- *Batch* størrelse: 16
- Læringsrate: 0.04
- Antall iterasjoner: 6000

SSD_mobilenet_v2, trening uten tomme veier

- Bilde *Input* Størrelse: 320x320

- *Batch* størrelse: 16
- Læringsrate: 0.08
- Antall iterasjoner: 6000

SSD_mobilenet_v1, trening med tomme veier

- Bilde *Input* Størrelse: 640x640
- *Batch* størrelse: 8
- Læringsrate: 0.08
- Antall iterasjoner: 16000

De komplette *pipeline.config* filene som er brukt for trening er vedlagt i Vedlegg C, D og E.
Etter en komplett trening kan man kjøre en evaluering av modellen med det samme *scriptet*. Denne evalueringen gir resultater for tap, mAP, samt *precision* og *recall*.
For eksportering av modellene ble det brukt `exporter\_main\_v2.py`. Denne eksporterer modellen på TensorFlow sitt `saved_model|` format.
For hvert utgangspunkt med forhåndstrente modeller ble det testet med ulike læringsrate, *batch* størrelse og antall iterasjoner for å optimalisere modellene. Det ble også gjort et forsøk med å inkludere *input* bilder som er *True Negative*.

4.1.5 Trening med YOLO

Treningen av YOLO foregikk ved bruk av Darknet. Dette ble gjort gjennom å klone og følge instruksjonene til AlexeyAB sitt GitHub Repository [53]. AlexeyAB, Alexey Bochkovskiy, er som nevnt i delkapittel 3.1.1 en av forfatterne bak den originale YOLOv4-rapporten [16]. Av den grunn, i tillegg til at *repositoriet* støttet trening av YOLOv4-Tiny-modellen, ble det derfor sett på som hensiktsmessig å ta utgangspunkt i hans *repository*. Darknet er originalt konfigurert for trening på COCO-datasettet, så endringer måtte gjøres for å kunne tilpasse nettverket dette studiets problem.

Først måtte datasettet legges inn, noe som ble gjort under `data/obj` i mappestrukturen. Dette medførte at innholdet i filer som `obj.names` og `obj.data` måtte endres. Filen `obj.names` inneholder navnet på klassene som er knyttet til objektene som er ment å detektere. Ettersom det kun er én klasse som detekteres i dette tilfellet, behøvde kun filen å inneholde *"car"*. Filen `obj.data` inneholder informasjon om hvor vektene skal lagres, samt filbanen til de to filene `train.txt` og `valid.txt`. I disse filene oppgis filbanen til hvert enkelt bilde som brukes under henholdsvis trening og validering. I Darknet blir det i tillegg til vekter fra hver iterasjon, lagret en fil med de beste vektene, hvor denne er basert på *Early Stopping*-teknikken. Dersom det oppstår problemer med *overfitting* underveis i treningen, kan man likevel benytte vekter som ikke er påvirket av dette.

Deretter måtte det opprettes en konfigurasjonsfil basert på en allerede eksisterende `.cfg`-fil for YOLOv4-Tiny. I Listing 4.2 listes de mest essensielle endringene som er gjort i både [net], [yolo], og [convolution]-lag før [yolo]-lagene. Denne konfigurasjonsfilen ble benyttet for trening både med og uten tomme veier. I dokumentasjonen til Darknet oppgis det at anbefalt antall iterasjoner settes lik antall klasser $\cdot$ 2000, men samtidig ikke lavere enn 6000. Grunnet at kun én klasse detekteres, ville verdien blitt lavere enn anbefalt. Dermed er 6000 brukt for antall iterasjoner, her kalt `max_batches`. Videre er `steps` definert som 80% og 90% av `max_batches`, noe som gir henholdsvis verdiene 4800 og 5400. I *[convolution]*-laget før hvert [yolo]-lag måtte `filter` endres til (klasser + 5) $\cdot$ 3, som her ga 18. Variabelen `random` i [yolo]-laget sørget for å endre størrelsen på nettverket for å trene på ulike oppløsninger.

```
[net]
max_batches = 6000
steps = 4800, 5400

[convolution]
filter = 18

[yolo]
classes = 1
random = 1
```

Listing 4.2: Tilpasninger gjort i konfigurasjonfilen

I tillegg ble det forsøkt å forbedre deteksjonen gjennom å endre tre variabler, som vist i Listing 4.3 i [yolo]-laget, da disse skulle gjøre *bounding* boksene mer nøyaktige. Denne konfigurasjonen ble også brukt på de to ulike scenarioene.

```
[yolo]
iou_normalizer = 0.5
iou_loss = giou
ignore_thres = 0.9
```

Listing 4.3: Tilpasninger gjort i konfigurasjonfilen for økt nøyaktighet

Konfigurasjonsfilen for YOLOv4-Tiny inneholder allerede parametre som kjører *data augmentation* automatisk under trening. Dette er som nevnt i delkapittel 2.2 fordelaktig i begrensede og lite varierte datasett.

Til slutt ble ferdigtrente vekter for valgt konfigurasjonsfil lastet ned og lagt i `data`-mappen. Ettersom konfigurasjonen til nettverket var tilpasset YOLOv4-Tiny, ble korresponderende vektfil `yolov4-tiny.conv.29` benyttet.

4.1.6 Deteksjon med SSD

For å visualisere deteksjonene til SSD ble det brukt kode basert på følgende [54]. Mindre endringer til koden ble gjort for kompatibilitet. Denne koden itererer gjennom en *input* video og kjører en deteksjon for hvert *frame*. Denne deteksjonen blir illustrert ved å ta i bruk en funksjon fra følgende kode "visualization_utils.py" fra TensorFlow Objekt Deteksjons APIet. Denne funksjonen henter ut koordinatene samt en verdi på hvor sikker modellen er for deteksjonen og tegner dette i *framet*. Hvert *frame* blir deretter skrevet til en ny video fil som viser et overblikk av hvordan modellen presterer på aktuell data.
I tillegg til dette er det gjort en visualisering av deteksjonene som blir gjort under evaluering av modellen. For dette blir følgende kode brukt "confusion_matrix_tf2.py" skrevet av Santiago Valdarrama [55]. Denne koden tegner opp både en boks for prediksjonen som blir gjort samt *ground truth* boksen som er definert i valideringsdatasettet. Ved å visualisere resultatene på denne måten kan man enklere luke ut hvilke tilfeller der modellen feiler og eventuelle gjentakende feil.

4.1.7 Deteksjon med YOLO

AlexeyAB sin GitHub [53] har i tillegg til trening av objektdetekteringsmodeller også støtte for å utføre selve deteksjonen. Det ble likevel valgt å benytte et annet rammeverk for denne delen, da det var ønskelig å kunne kjøre deteksjon på en mindre, innebygd enhet med kun støtte for TensorFlow Lite. Alexey refererer til et annet GitHub *repository* [56] som tilbyr nettopp dette. I dette prosjektet har et *repository* av The AI Guy [57], en *fork* av Viet Hùngs *repository* [56], i stedet blitt tatt i bruk. Hovedårsaken til dette at prosjektet hadde en detaljert fremgangsmåte som var nyttig i startfasen av prosjektet, samt at det inneholdt en del ekstra funksjoner.

For å kunne gjøre deteksjoner med trente vekter fra Darknet, måtte disse først konverteres til TensorFlow. Ettersom det var ønskelig å detektere på eget test-datasett, ble det lagt inn en mappe med test-bilder under `data`. Originalt var det kun mulig å kjøre deteksjon på flere bilder ved å sende inn filbanen til disse som argument på formen "./filbane/bilde1.jpg, ./filbane/bilde2.jpg, ...". Ettersom testdatasettet består av 176 bilder, ble det sett på som uhensiktsmessig å sende inn bildene på denne måten. Derfor ble koden endret til å lese en fil med filbanen til ønskede testbilder, tilsvarende `train.txt` og `valid.txt` som benyttes i Darknet. I tillegg til å vise de predikerte *bounding* boksene, var det ønskelig å sammenligne disse mot *ground truth* boksene. Dette ble gjort gjennom å legge inn egen kode for lesing og tegning av filer med de opprinnelige koordinatene. For å få en indikasjon på hvordan deteksjonen ville foregå i sanntid, ble modellene også testet på video med de samme bildene fra test-dataen.

4.1.8 Sporing med YOLO

Underveis i prosjektet ble det trente YOLO-modellene testet på sporingsalgoritmen DeepSORT. TheAIGuy, forfatteren bak *repositoriet* for konvertering fra Darknet-vekter til TensorFlow nevnt i delkapittel 4.1.7, har i tillegg et rammeverk for YOLOv4-Tiny i kombinasjon med DeepSORT [58]. Sporing ble som nevnt tidligere sett på som en sekundærprioritet til deteksjon, da god deteksjon er fundamentalt for god sporingen. Det ble dog sett på som aktuelt å validere modellenes deteksjonprestasjon underveis ved å utføre sporing med YOLO-vekter vekter på en testvideo.

4.2 Teknologi

4.2.1 Biblioteker og verktøy

TensorFlow

TensorFlow er et *open source* bibliotek, utviklet av Google, for bygging av maskinlæringsmodeller [59]. Biblioteket har støtte for kjøring på både CPU og GPU. For å laste ned TensorFlow kan man bruke pip som er en *package manager* for Python.

OpenCV

Open Source Computer Vision Library (OpenCV) er et *open source* bibliotek som er sentralt innen CV [60], og har i dette prosjektet blitt benyttet både i prosessen før og under trening av modell. OpenCV har grensesnitt for flere programmeringsspråk, for eksempel C++ og Python. For bruk av OpenCV i Darknet krevde dette nedlasting av biblioteket med C++-grensesnittet gjennom de to GitHub-repoene [61] og [62], som inneholder moduler for ekstra funksjonalitet. For SSD ble OpenCV installert/bygd på Windows ved hjelp av programmet CMake. OpenCV er ellers i prosjektet implementert med Python-grensesnittet.

CUDA og cuDNN

Compute Unified Device Architecture (CUDA) er en plattform for parallell databehandling, utviklet av NVIDIA, som gjør det mulig å eksekvere programkode på GPU. Med CUDA har utnyttelsen av GPU ført til raskere utførelse av dataprogrammer , noe som blant annet er fordelaktig i deep learning-problemet i dette prosjektet. [63] Minimumskrav for at denne programvaren skal virke er at man har et nytt og kraftig nok skjermkort, og laster ned CUDA-versjonen som passer det spesifikke grafikkortet. I dette prosjektet har CUDA-versjonen 10.1 og 11.1 blitt benyttet.

NVIDIA CUDA Deep Neural Network, forkortet til cuDNN, er et bibliotek for dype nevrale nettverk (DNN) for optimalisering av GPU-akselerasjonen [64]. Denne programvaren er også delt inn i versjoner, og versjonen man benytter seg av må være tilpasset CUDA-versjonen. I dette prosjektet anvendes cuDNN-versjonene v7.6.5, v8.0.4, og v8.1.0.

Idun

Idun er et kluster som tilbys til ansatte og studenter ved NTNU for å utføre større beregninger gjennom ressurser som CPU og GPU [65, 66]. Håndteringen av ressurser og jobber gjøres av Slurm Workload Manager, også kjent som SLURM [67]. Gjennom et SLURM *Job Script* settes nødvendige ressurser for å utføre en jobb opp, og basert på disse opplysningene blir man satt i en prioritert kø. Det betyr at forspørsel om færre ressurser vil gi en høyere prioritet i køen, og det er generelt anbefalt å ikke spørre etter flere ressurser enn nødvendig. *Scriptet* må også inneholde opplysninger om nødvendige moduler (softvare) for å kunne utføre ønsket jobb.

4.2.2 Maskinvare

I dette prosjektet har det blitt brukt ulike maskiner for trening og detektering for de to objektdetekteringsmodellene SSD og YOLO. For trening av SSD har en av gruppemedlemmenes maskin blitt benyttet. Spesifikasjonene kan ses i Tabell 4.2

	CPU	GPU
1	AMD Ryzen 5 3600X	NVIDIA GeForce RTX 3070

Tabell 4.2: Maskinvare brukt under trening og deteksjon på SSD-modellene

Trening av YOLO-modellene har hovedsakelig foregått på Idun [65, 66] klusteret, hvor vi har fått tilgang på GPU-*noder* med to ulike spesifikasjoner gitt i Tabell 4.3.

	CPU	GPU
1	Intel Xeon E5-2650v4	NVIDIA Tesla P100
2	Intel Xeon Gold 6148	NVIDIA Tesla V100

Tabell 4.3: Maskinvare brukt under trening av YOLO-modellene

Ettersom det var problemer med å utføre deteksjonsprosessen for YOLO på videofiler på Idun-clusteret, måtte en av gruppemedlemmenes maskin tas i bruk. Spesifikasjonene til denne maskinen er gitt i Tabell 4.4. For deteksjon med SSD-modellene, er maskinen med spesifikasjonene gitt i Tabell 4.2 brukt.

	CPU	GPU
1	Intel i7-4790K	NVIDIA GeForce GTX 970

Tabell 4.4: Maskinvare brukt ved deteksjon på YOLO-modellene

5 Resultater

Dette kapittelet presenterer de oppnådde resultatet til de ulike objektdetekteringsmodellene som er undersøkt i prosjektet. Som følge av at det har blitt benyttet to datasett til trening og validering, vil resultatene til hver av disse presenteres i hvert sitt delkapittel. I tillegg til resultatene fra trening, vil også modellenes prestasjon på testsettet presenteres og visualiseres.

Totalt har sju objektdetekteringsmodeller for deteksjon av biler blitt testet, tre på datasettet med tomme veier og fire på datasettet uten tomme veier. På datasettet med tomme veier har én SSD-modell og to YOLO-modeller blitt trent og validert. På datasettet uten tomme veier har trening og validering foregått på to SSD- og to YOLO-modeller. For YOLO, som beskrevet i delkapittel 4.1.5, ble det utarbeidet én versjon med standard konfigurasjon, samt én med tilpasninger som skulle forbedre *bounding* boksene. Disse vil heretter refereres som henholdsvis YOLO_{DEF} og YOLO_{OPT}. Videre vil også modellene navngis ut i fra hvilket datasett de er trent på: MT (Med Tomme veier) og UT (Uten Tomme veier).

5.1 Trening og validering med tomme veier

Tabell 5.1 presenterer tapet og de to mAP-verdiene, mAP@50 og mAP@75, for de ulike modellene trent og validert med tomme veier. Prestasjonen til $\text{YOLO}_{\text{DEF, MT}}$ og $\text{YOLO}_{\text{OPT, MT}}$, som presenteres i tabellen, er basert på de beste vektene som ble oppnådd. Disse kom etter henholdsvis 6000 iterasjoner og 3000 iterasjoner. $\text{SSD}_{\text{MobileNet\_V1}}$ oppnådde de beste vektene etter 16000 iterasjoner. For datasettet hvor tomme veier er inkludert ble det testet med totalt tre modeller, én SSD-modell og to YOLO-modeller. Tilpasningene for hver av disse modellene er beskrevet i kapittel 4.1.4 og 4.1.5. I Tabell 5.2 er fordelingen av antall *True Positive* (TP), *False Positive* (FP) og *False Negative* (FN), samt tilhørende *Precision* (PR), *Recall* (RE) og F1-*score*, oppført for mAP@50 og mAP@75 for hver av modellene.

Modell	Tap	mAP@50	mAP@75
$\text{SSD}_{\text{MobileNet\_v1, MT}}$	1.91	0.90	0.75
$\text{YOLO}_{\text{DEF, MT}}$	0.05	1.00	0.96
$\text{YOLO}_{\text{OPT, MT}}$	0.12	0.99	0.95

Tabell 5.1: Prestasjonen til de ulike modellene trent og validert med tomme veier, målt i tap, mAP@50 og mAP@75

Modell	mAP	TP	FP	FN	PR	RE	F1
$SSD_{MobileNet\_v1,MT}$	mAP@50	785	28	59	0.97	0.93	0.95
$YOLO_{DEF,MT}$	mAP@50	838	35	6	0.96	0.99	0.98
	mAP@75	816	57	28	0.93	0.97	0.95
$YOLO_{OPT,MT}$	mAP@50	812	30	32	0.96	0.96	0.96
	mAP@75	785	57	59	0.93	0.93	0.93

Tabell 5.2: Oversikt over antall TP, FP og FN, samt tilhørende PR, RE og F1-score for $SSD_{MobileNet\_v1,\ MT}$, standard YOLO ($YOLO_{DEF}$) og forbedret YOLO ($YOLO_{OPT}$) av totalt 844 biler, oppgitt for både mAP@50 og mAP@75. Modellene er trent og validert på datasettet med tomme veier, derav MT

Både $YOLO_{DEF,\ MT}$ og $YOLO_{OPT,\ MT}$ oppnår høy nøyaktighet og lave verdier for tap (Tabell 5.1). mAP@50 for de nevnte modellene er tilnærmet like, men for mAP@75 har den tilpassede YOLO-modellen, $YOLO_{OPT,\ MT}$, en nedgang på 1%. Ut fra Tabell 5.2 kan det observeres at det er en reduksjon av antall TP, og økning i antall FP og FN ved mAP@75 for både $YOLO_{DEF,\ MT}$ og $YOLO_{OPT,\ MT}$. Dette gir utslag gjennom en reduksjon på 3% for F1-score for de to modellene ved mAP@75, men sett i sammenheng med delkapittel 2.4.1 er verdiene tilfredsstillende. For $SSD_{MobileNet\_v1,\ MT}$ observeres et noe høyere tap sammenlignet med de to YOLO-modellene, samt en lavere verdi for både mAP@50 og mAP@75. Til tross for dette viser resultatene at modellen oppnår høye verdier for PR, RE og F1-score.

5.2 Trening og validering uten tomme veier

For datasettet uten tomme veier ble det testet med totalt fire modeller, to SSD-modeller og to YOLO-modeller. Tabell 5.3 presenterer tapet og de to mAP-verdiene, mAP@50 og mAP@75, for de ulike modellene trent og validert uten tomme veier. Prestasjonen til $YOLO_{DEF,\ MT}$ og $YOLO_{OPT,\ MT}$ som presenteres i tabellen, er basert på de beste vektene som ble oppnådd. Disse kom etter henholdsvis 6000 iterasjoner og 4000 iterasjoner. $SSD_{MobileNet\_v1}$ og $SSD_{MobileNet\_v2}$ oppnådde de beste vektene etter 6000 iterasjoner. Fordelingen av antall *True Positive* (TP), *False Positive* (FP) og *False Negative* (FN), samt tilhørende *Precision* (PR), *Recall* (RE) og F1-*score*, oppført for mAP@50 og mAP@75 for hver av modellene. kan ses i Tabell 5.4.

Modell	Tap	mAP@50	mAP@75
$SSD_{MobileNet\_v1,\ UT}$	0.84	0.98	0.96
$SSD_{MobileNet\_v2,\ UT}$	0.23	0.98	0.93
$YOLO_{DEF,UT}$	0.05	1.00	0.97
$YOLO_{OPT,UT}$	0.16	0.99	0.94

Tabell 5.3: Prestasjonen til de ulike modellene trent og validert uten tomme veier, målt i tap, mAP@50 og mAP@75

Modell	mAP	TP	FP	FN	PR	RE	F1
$SSD_{MobileNet\_v1,UT}$	mAP@50	835	19	9	0.98	0.99	0.98
$SSD_{MobileNet\_v2,UT}$	mAP@50	810	25	23	0.97	0.96	0.97
$YOLO_{DEF,UT}$	mAP@50	837	25	7	0.97	0.99	0.98
	mAP@75	821	43	23	0.95	0.97	0.96
$YOLO_{OPT,UT}$	mAP@50	810	21	34	0.97	0.96	0.97
	mAP@75	782	49	62	0.94	0.93	0.93

Tabell 5.4: Oversikt over antall TP, FP og FN, samt tilhørende PR, RE og F1-score for $SSD_{MobileNet\_v1,\ UT}$, $SSD_{MobileNet\_v2,\ UT}$ standard YOLO ($YOLO_{DEF}$) og optimalisert YOLO ($YOLO_{OPT}$), av totalt 844 biler, oppgitt for både mAP@50 og mAP@75. Modellene er trent og validert på datasettet uten uten tomme veier, derav UT

Fra Tabell 5.3 observeres svært tilfredsstillende resultater for de to SSD-modellene og de to YOLO-modellene. Felles for de fire modellene er at de har et lavt tap, samt høye verdier for både mAP@50 og mAP@75. YOLO-modellen med standard konfigurasjoner, $YOLO_{DEF,\ UT}$, oppnår lavest tap, og høyeste verdier for mAP@50 og mAP@75 med henholdsvis 100% og 97%. Til sammenligning er disse verdiene 2% og 1% høyere enn tilsvarende verdier for $SSD_{MobileNet\_v1,\ UT}$, som presterer best av SSD modellene. Ut ifra Tabell 5.4 kan det observeres svært like resultater for $YOLO_{DEF,\ UT}$ og $YOLO_{OPT,\ UT}$ som for YOLO-modellene trent med tomme veier (Tabell 5.2). For $SSD_{MobileNet\_v1,\ UT}$ og $SSD_{MobileNet\_v2,\ UT}$ kan det observeres høyere verdier for TP, samt lavere verdier for FN og FP, sammenlignet med SSD-modellen trent med tomme veier, $SSD_{MobileNet\_v1,\ MT}$ (Tabell 5.2).

5.3 Deteksjon

I Tabell 5.5 presenteres fordelingen av antall *True Positive* (TP), *False Positive* (FP) og *False Negative* (FN), samt tilhørende *Precision* (PR), *Recall* (RE) og F1-*score*, oppført for mAP@50 for SSD- og YOLO-modellene som er benyttet i dette prosjektet. Deteksjonene på både bilder og video er gjort med mAP@50, da 0.5 er en standard verdi for IoU-grense i forbindelse med detektering (delkapittel 2.4.1).

Modell	TP	FP	FN	PR	RE	F1
$SSD_{MobileNet\_v1,\ MT}$	161	1	25	0.99	0.87	0.93
$SSD_{MobileNet\_v1,\ UT}$	183	7	3	0.96	0.98	0.97
$SSD_{MobileNet\_v2,\ UT}$	176	4	10	0.98	0.95	0.97
$YOLO_{DEF,\ MT}$	173	3	13	0.98	0.93	0.96
$YOLO_{OPT,\ MT}$	133	9	53	0.94	0.72	0.83
$YOLO_{DEF,\ UT}$	179	3	7	0.98	0.96	0.97
$YOLO_{OPT,\ UT}$	133	3	53	0.98	0.72	0.85

Tabell 5.5: Oversikt over fordeling av antall TP, FP og FN, samt tilhørende PR, RE og F1-score på testsettet med totalt 186 biler

Som man kan observere i Tabell 5.5 er det svært like resultater for de to $YOLO_{DEF}$-modellene. Tilsvarende gjelder også for de to $YOLO_{OPT}$-modellene. Sammenlignet med resultatene fra Tabell 5.2 og Tabell 5.4, hvor det var liten differanse mellom *Precision*-, *Recall*- og følgelig F1-verdiene for de ulike modellene, kan man i Tabell 5.5 se en tydeligere forskjell blant $YOLO_{DEF}$- og $YOLO_{OPT}$-modellene. Årsaken til dette er en markant økning i antall FN både for $YOLO_{OPT,\ MT}$ og $YOLO_{OPT,\ UT}$. Basert på opplysningene i Tabell 5.5 er $SSD_{MobileNet\_v1,\ UT}$ modellen som oppnådde de beste resultatene, med høyeste antall TP, samt lavest antall FN.

I forbindelse med resultatene presentert i Tabell 5.5, er det valgt ut fire scenarier for visualisering. De presenterte scenarioene er eksempler på kritiske situasjoner som kan oppstå. Interessante problemstillinger er for eksempel hvordan modellene tolker kjøretøy som lastebiler, hvor godt modellene klarer å detektere biler som befinner seg lengst vekk fra kamera, hvordan enkelte lysforhold kan ha innvirkning på deteksjonene, samt hvordan modellene presterer når biler kjører i parallel. De ulike casene kan ses i henholdsvis Figur 5.1, Figur 5.2, Figur 5.3 og Figur 5.4. Som man kan observere i figurene, er $YOLO_{DEF,\ UT}$ modellen som presterer best på de ulike scenariene. Modellen feilklassifiserer ikke lastebilen som en bil, detekterer bilen som befinner seg lengst vekk fra kamera, samt bilen i vanskelige lysforhold. Modellen klarte også å detektere og skille bilene som kjørte i parallell.

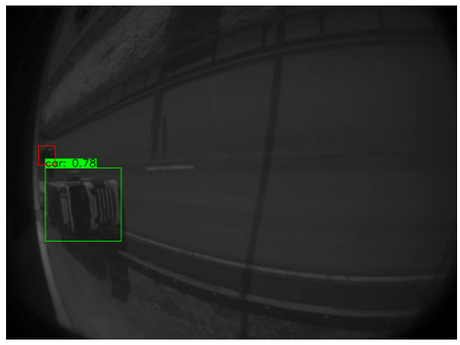
Video av deteksjonen til de ulike modellene kan ses i Vedlegg B.

Scenario 1: Bil og lastebil

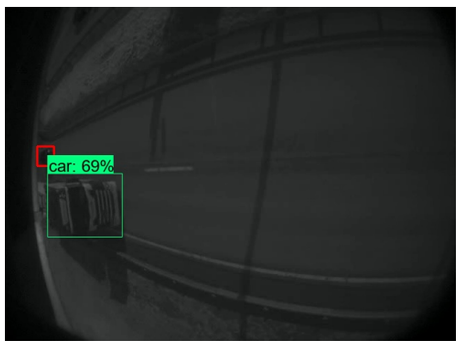
(a) $\text{YOLO}_{\text{DEF, MT}}$

(b) $\text{YOLO}_{\text{OPT, MT}}$

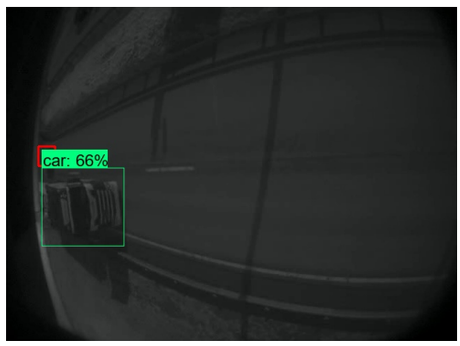
(c) $\text{YOLO}_{\text{DEF, UT}}$



(d) $\text{YOLO}_{\text{OPT, UT}}$



(e) $\text{SSD}_{\text{MobileNet\_v1, MT}}$

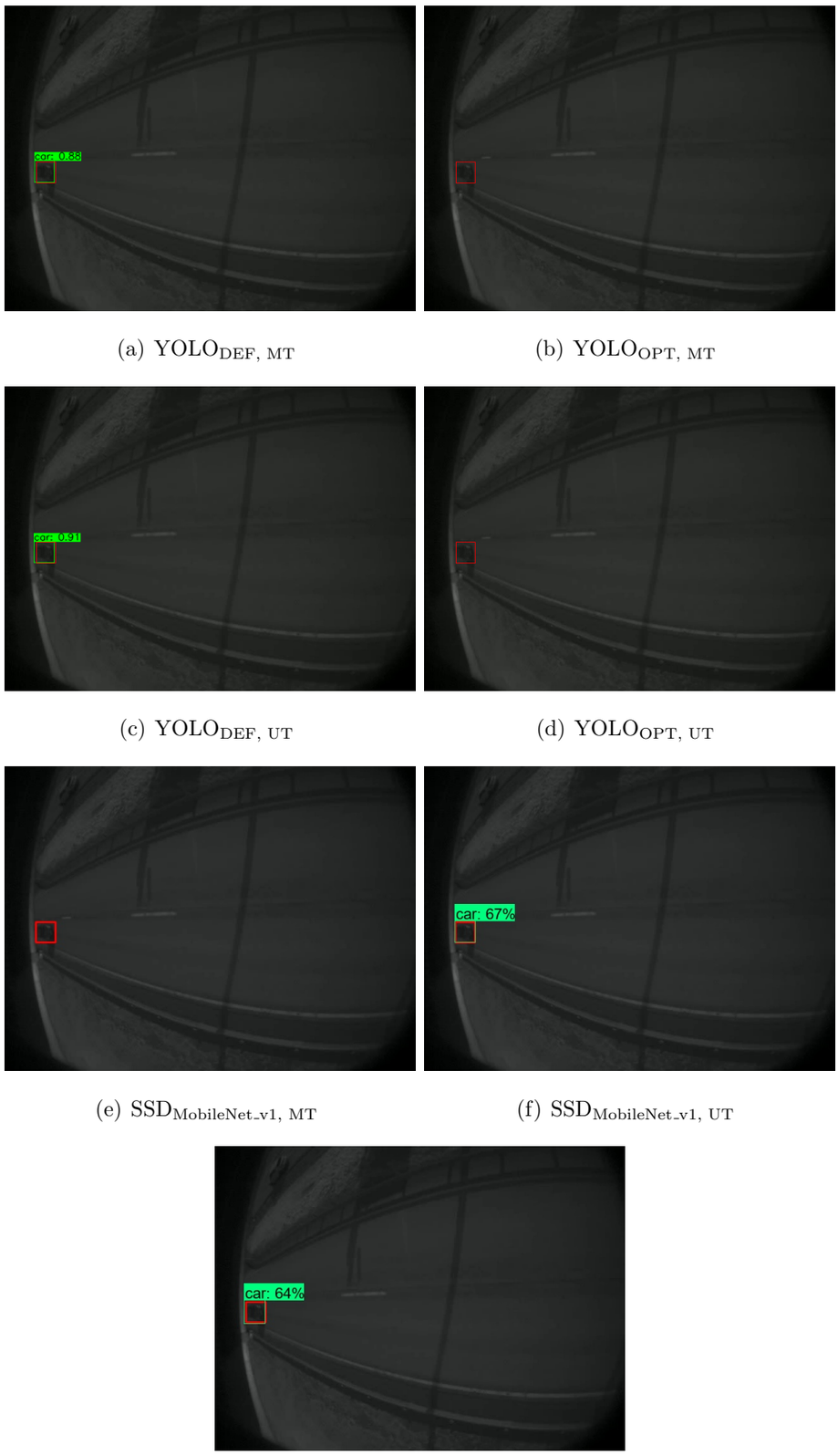


(f) $\text{SSD}_{\text{MobileNet\_v1, UT}}$

(g) $\text{SSD}_{\text{MobileNet\_v2, UT}}$

Figur 5.1: Eksempel på deteksjon med bil og lastebil i frame. Deteksjoner presenteres som grønne rektangler med klasse og *confidence score*, og røde rektangler indikerer ground truth

Scenario 2: Små biler



(a) $\text{YOLO}_{\text{DEF, MT}}$

(b) $\text{YOLO}_{\text{OPT, MT}}$

(c) $\text{YOLO}_{\text{DEF, UT}}$

(d) $\text{YOLO}_{\text{OPT, UT}}$

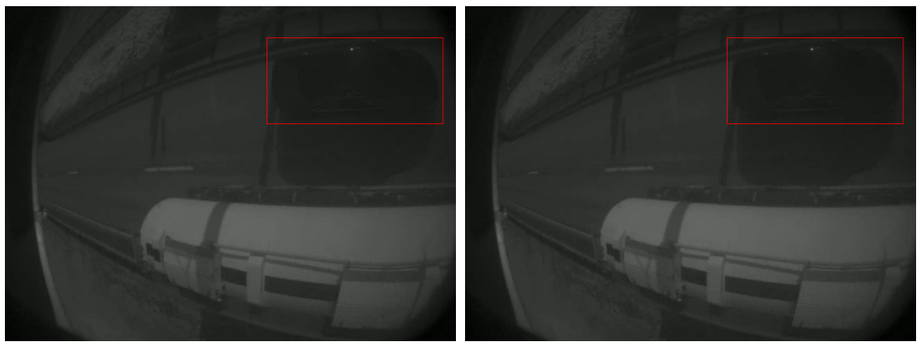
(e) $\text{SSD}_{\text{MobileNet\_v1, MT}}$

(f) $\text{SSD}_{\text{MobileNet\_v1, UT}}$

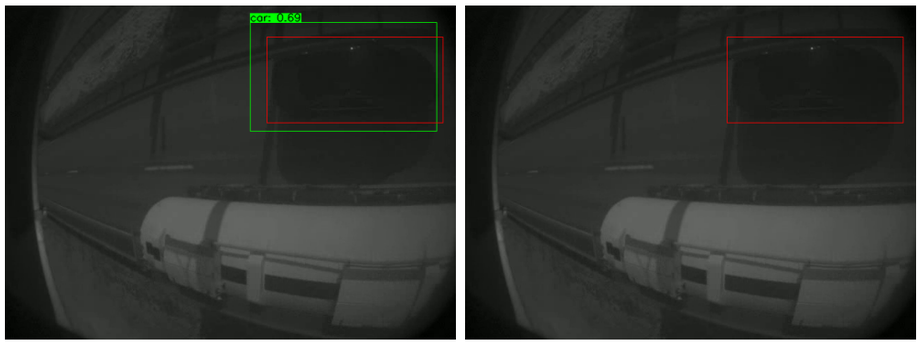
(g) $\text{SSD}_{\text{MobileNet\_v2, UT}}$

Figur 5.2: Eksempel på deteksjon med bil på vei inn i bomstasjon. Deteksjoner presenteres som grønne rektangler med klasse og *confidence score*, og røde rektangler indikerer ground truth

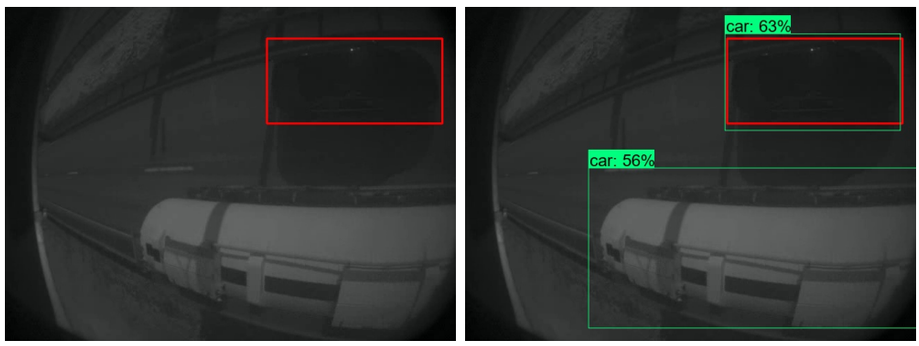
Scenario 3: Vanskelige lysforhold



(a) $\text{YOLO}_{\text{DEF, MT}}$ (b) $\text{YOLO}_{\text{OPT, MT}}$



(c) $\text{YOLO}_{\text{DEF, UT}}$ (d) $\text{YOLO}_{\text{OPT, UT}}$



(e) $\text{SSD}_{\text{MobileNet\_v1, MT}}$ (f) $\text{SSD}_{\text{MobileNet\_v1, UT}}$

(g) $\text{SSD}_{\text{MobileNet\_v2, UT}}$

Figur 5.3: Eksempel på deteksjon med mørk bil i skygge. Deteksjoner presenteres som grønne rektangler med klasse og *confidence score*, og røde rektangler indikerer ground truth

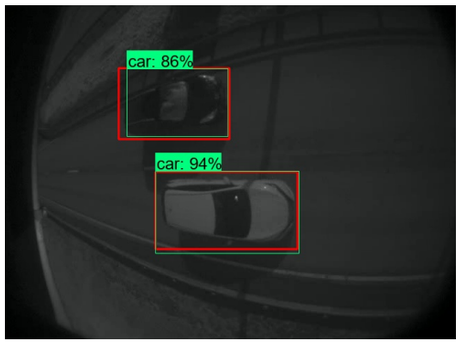
Scenario 4: Biler i parallell

(a) $\text{YOLO}_{\text{DEF, MT}}$

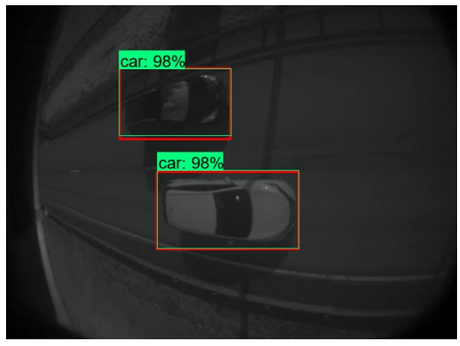
(b) $\text{YOLO}_{\text{OPT, MT}}$

(c) $\text{YOLO}_{\text{DEF, UT}}$

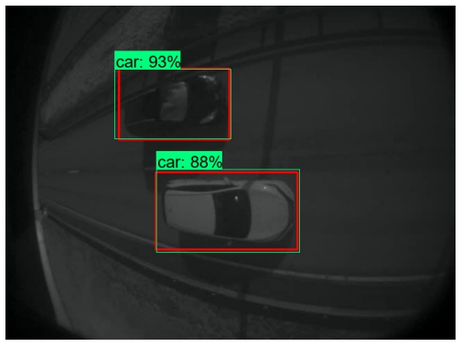
(d) $\text{YOLO}_{\text{OPT, UT}}$



(e) $\text{SSD}_{\text{MobileNet\_v1, MT}}$



(f) $\text{SSD}_{\text{MobileNet\_v1, UT}}$

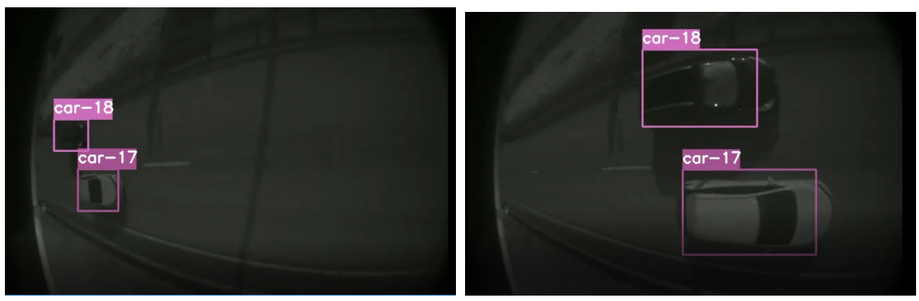


(g) $\text{SSD}_{\text{MobileNet\_v2, UT}}$

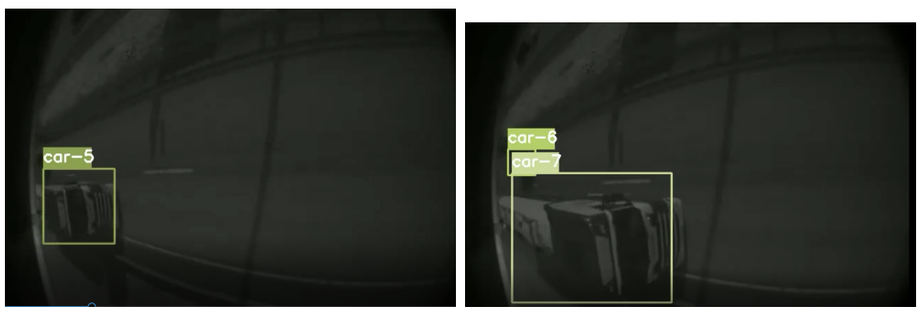
Figur 5.4: Eksempel på deteksjon med to biler i parallell. Deteksjoner presenteres som grønne rektangler med klasse og *confidence score*, og røde rektangler indikerer ground truth

5.4 Sporing med YOLOv4-Tiny og DeepSORT

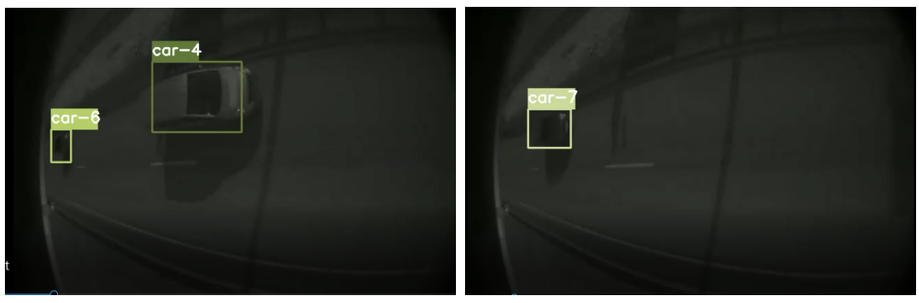
Video av sporing kan ses i Vedlegg B



Figur 5.5: Eksempel på to biler med korrekt sporing, $\text{YOLO}_{\text{DEF, UT}}$



Figur 5.6: Eksempel på sporeproblem med lastebil, $\text{YOLO}_{\text{DEF,UT}}$



Figur 5.7: Eksempel på ID-svitsjing fra 6 til 7 på bakre bil. Ingen bil tildelt ID lik 5, $\text{YOLO}_{\text{DEF, MT}}$

6 Diskusjon

I dette kapittelet gjøres det rede for resultatene fra kapittel 5, og disse vil deretter drøftes og vurderes ut ifra de stilte forskningsspørsmålene fra delkapittel 1.2.

Hensikten med oppgaven er å finne alternativer til gode objektdeteksjonsalgoritmer for å detekte biler i bildeframes. Algoritmene YOLO og SSD ble valgt på bakgrunn av studiet som ble gjort i starten av prosjektet. Utvikling av disse modellene ble arbeidet med i parallel med fokus på å oppnå likest mulig utgangspunkt for sammenligning. YOLO- og SSD-modellene som ble testet er algoritmene YOLOv4-Tiny og SSD MobileNet.

Tabell 5.1 og Tabell 5.2 viser resultatene etter evaluering av modellene trent på datasettet inkludert tomme veier. Alle modellene viser akseptable resultater med en mAP@50 på over 90% og en mAP@75 på over 75%. YOLO-modellene presterer derimot noe bedre enn SSD-modellen der den beste YOLO-modellen $\text{YOLO}_{\text{DEF, MT}}$ har mAP@50 på 100% og mAP@75 på 96%. De oppnådde YOLO-resultatene er betydelig bedre enn YOLOv4-Tiny sin målte mAP på 38.1% (Tabell 3.1). Dette gjelder også for SSD-modellen, der den forhåndtrente modellen ssd_mobilenet_v1_fpn_640x640 har en mAP på 29.1% og modellen ssd_mobilenet_v2_320x320 har en mAP på 20.2% på COCO datasettet. Antageligvis er årsaken at modellene i dette prosjektet kun er trent på én klasse, mens YOLOv4-Tiny og $\text{SSD}_{\text{MobileNet\_v1}}$, som nevnt i delkapittel 3.1.2 og 4.1.4, er trent på COCO-datasett bestående av 80 ulike klasser. I dette studiet vil et detektert objekt kun klassifiseres som en bil, og derfor ikke bli straffet i like stor grad som i tilfeller hvor det skal skilles mellom flere klasser.
$\text{SSD}_{\text{MobileNet\_v1}}$ modellen krevde et større antall iterasjoner enn YOLO for å oppnå de beste vektene når tomme veier er inkludert i datasettet. SSD og YOLO er trent på ulik maskinvare (delkapittel 4.2.2) der treningen av SSD har hatt større begrensinger for både *batch* størrelse og treningshastighet. $\text{SSD}_{\text{MobileNet\_v1}}$ er i tillegg trent på *input* bilder med større bildestørrelse enn YOLO-modellene. Dette sammen med maskinvaren førte til en lav trening *batch* størrelse for $\text{SSD}_{\text{MobileNet\_v1}}$, som kan ha hatt en påvirkning på antall iterasjoner.

Tabell 5.3 og Tabell 5.4 viser resultatene etter evalueringen av modellene trent på datasettet uten tomme veier. Disse er i likhet med resultatene presentert i Tabell 5.1 og 5.2 gode resultater. Jevnt over presterer både YOLO og SSD modellene bedre med trening på dette datasettet. Her har alle modellene en mAP@50 verdi på over 98% og en mAP@75 verdi på over 93%. Man kan også se en reduksjon i antall FP og FN prediksjoner sammenlignet med resultatene fra modellene trent med de tomme veiene. Spesielt gjelder dette $\text{SSD}_{\text{MobileNet\_v1}}$. En årsak til dette kan være at modellene som er trent med tomme veier oftere detekterer FP og FN fordi datasettet består av flere bilder som øker mulighet for feilklassifisering. Derimot består disse bildene kun av

tomme veier som i teorien skulle forbedret modellens evne til å skille et objekt fra omgivelsene. I datasettet med tomme veier, er i underkant halvparten av datasettet bilder av tomme veier. Det er mulig at den dårlige deteksjonsraten kommer av at bildene med *lablet* bilobjekter har mindre påvirkning på justeringen av vektene under trening, da de tar opp mindre av datasettet. Da vil en reduksjon av antall bilder med tomme veier kunne forbedre resultatet.

Alle modellene viser gode resultater etter evaluering, men visualisering av deteksjonene i Figur 5.1-5.4 viser at modellene fortsatt gjør kritiske feil i prediksjonene, som vil skape problemer for en sporingsalgoritme. Et av de mest sentrale problemene er at modellene kun er trent på personbiler. Modellene er dermed ikke kjent med andre typer kjøretøy og har en tendens til å identifisere også disse som personbiler. Et eksempel på dette kan ses i Figur 5.2(a-f). Problemet oppstår i hovedsak når ikke-klassifiserte kjøretøy er på vei inn i *frame*, når kun fronten vises. Dette i tillegg til kameravinkelen som bildene er tatt og lysforholdene i enkelte tilfeller fører til at de distinkte forskjellene mellom ulike kjøretøy blir mindre og dermed vanskeligere for modellene å skille. For eksempel kan fronten til en lastebil bli forvekslet med en personbil. Det er sannsynlig at et mer variert datasett med flere klasser for å skille mellom ulike typer kjøretøy vil være løsning på dette problemet. Flere klasser vil derimot øke sannsynligheten for feilklassifisering, da det stilles høyere krav til modellenes evne til å trekke ut og identifisere sentrale *features* for hver av de inkluderte klassene. Dersom det er færre klasser å skille mellom, er sannsynligheten for at modellen klassifiserer korrekt større. Treningen og evalueringen som er gjort i dette prosjektet baserer seg på et datasett hovedsakelig bestående av personbiler. Dette fører til at klassen "bil" sin representasjon i datasettet nærmer seg 100%. Her vil en riktig klassifisering være mer sannsynlig fordi alle deteksjoner vil klassifiseres som bil, som vil være rett for nesten alle tilfeller i datasettet. Om flere klasser er inkludert ville datasettet ideelt være jevnt fordelt på klassene, og sannsynligheten for riktig klassifisering vil gå ned.

Et stort nok datagrunnlag er derimot ikke tilgjengelig for andre klasser enn personbil. Dette skyldes at all data som er brukt er manuelt *lablet* og at videofilene som Q-Free har tilbudt primært består av personbiler. I tillegg er bilene i datasettet avbildet fra et fugleperspektiv, som er en annen vinkel enn det utenforstående datasett stortsett bruker. I utenforstående datasett som for eksempel COCO er bilene stortsett avbildet i normalperspektiv. Det som da er kategorisert som en bil i dette prosjektet sitt datasett vil dermed ikke stemme overens med det de forhåndstrente modellene er trent på. Å hente data fra andre kilder er heller ikke mulig, som total skaper en begrensing for mengden data som er tilgjengelig for trening.

Overfitting er som nevnt i delkapittel 2.2 et problem som kan oppstå under trening av nevrale nettverk. Man kan se antydning til *overfitting* også i dette prosjektet, da de

beste vektene for begge $YOLO_{OPT}$-modellene kom fra tidlige iterasjoner. Dette kan være en av grunnene til at det tilsynelatende er god presisjon på $YOLO_{OPT}$-modellene under trening, men som ved videre testing viser seg å prestere dårligere (Tabell 5.5). Et av tiltakene som ble gjort i forsøk på å hindre *overfitting*, var som beskrevet i delkapittel 4.1.5 å ta utgangspunkt i vekter fra *Early Stopping*-punktet. Som man kan se i Tabell 5.2 og 5.4, ble antall TP og FP noe redusert sammenlignet med $YOLO_{DEF}$. Den største forskjellen kan ses gjennom antall FN, situasjoner hvor eksisterende biler ikke blir detektert. Denne verdien øker for $YOLO_{OPT}$-modellene, både med evaluering på mAP@50 og mAP@75. Problemet med *overfitting* og økningen i antall FN for nevnte modeller kommer spesielt til syne gjennom deres prestasjon på video (Vedlegg B) og test-datasettet (Tabell 5.5).

Ved en gjennomgang av både valideringss- og testsettet, ble det funnet eksempler hvor biler ikke hadde blitt *lablet*. Dette er ikke gunstig i noen av tilfellene, da det kan ha en påvirkning på prestasjonen til modellene. Tilfeller hvor biler ikke er markert, er i hovedsak på svært små biler på vei inn i bomstasjonen, samt biler på vei ut. Her syns kun fronten eller bakdelen av en bil, noe som gjør det vanskelig for oss å si med stor sikkerhet at objektet kan klassifiseres som en bil. Problemene knyttet til situasjoner hvor dataen ikke er korrekt markert, har størst påvirkning i treningen av modellen. Dersom modellene blir trent på data som er markert feil, vil dette påvirke modellenes nøyaktighet. Sett i sammenheng med delkapittel 2.2, vil en modell som detekterer en bil som ikke er markert ha en negativ effekt på sluttresultatet. Dette problemet kunne mulig blitt løst ved å inkludere deteksjon og klassifisering av bildeler og ikke bare hele biler, slik det var tenkt i den originale oppgaven. En slik løsning var derimot ikke aktuelt for dette prosjektet da manuell klassifisering av datasettet var nødvendig.

Deteksjon har vært hovedfokuset i dette prosjektet, men det er også testet med sporing ved bruk av YOLO og DeepSORT for å sjekke hvilke resultat som kan forventes. Som beskrevet i delkapittel 3.3.1, er prestasjonen på sporingen sterkt knyttet til deteksjonen. Basert på de oppnådde mAP-verdiene for deteksjon, samt fordeling av antall TP, FP og FN, burde $\text{YOLO}_{\text{DEF, MT}}$ og $\text{YOLO}_{\text{DEF, UT}}$ ha et relativ likt grunnlag for sporing. I situasjoner hvor $\text{YOLO}_{\text{DEF, UT}}$ klarer å identifisere samme objekt over flere *frames*, har $\text{YOLO}_{\text{DEF, MT}}$ problemer med ID-svitsjing (Figur 5.7). Begge modellene har derimot problemer når et ikke-klassifisert objekt kjører inn i *frame* 5.7. En bedre visualisering av dette kan ses i sporingsvideoene til de nevnte YOLO-modellene i Vedlegg B. Hvorfor det er forskjell mellom modellene er vanskelig å tolke ut ifra nåværende resultater.

7 Konklusjon og videre arbeid

I dette kapittelet trekkes først en konklusjon av arbeidet ut fra stilte forskningsspørmål, før områder som kan forbedres i videre arbeid diskuteres.

7.1 Konklusjon

Målet med prosjektet var å undersøke om maskinlæring og objektdeteksjon ved bruk av modellene YOLO og SSD gir gode nok resultater til at de kan brukes til sporing av bilobjekter gjennom en bomstasjon. Både YOLO og SSD modellene ga høy presisjon, der den modellen som presterte best på både trening og testing var $\text{YOLO}_{\text{DEF,UT}}$ med en mAP på tilnærmet 100% og F1 på 98% ved evalueringen. De andre modellene oppnådde også gode resultater, hvor det kun er en liten prosentvis differanse. Til tross for disse gode resultatene vil det være nødvendig å inkludere flere klassser for ulike kjøretøystyper om modellen skal klare å oppnå pålitelig sporing i kombinasjon med en sporingsalgoritme. Scenarier der algoritmene feiler er når lastebiler blir klassifisert som biler og når bilene i bildet er små og har utydelige kjennetegn. Totalt sett presterer YOLO og SSD svært likt for dette tilfellet og eventuelle forskjeller vil tydeliggjøres ved videre arbeid. Det er dermed grunn til å tro at valg av modell for videre arbeid ikke vil sette begrensninger for sluttresultatet og at et valg mellom YOLO og SSD i størst grad vil baseres på tidligere arbeid samt egen personlig preferanse.

7.2 Videre arbeid

For videre arbeid er det aktuelt å se på hvordan objektdeteksjonen kan forbedres. En mulig forbedringer som kan utforskes er implementasjon av flere kjøretøyklasser som objektdeteksjon modellen kan trenes på. Et alternativ eller tillegg til dette kan være å se på muligheten for å detektere bildeler slik at prediksjonene også er gode for bilder der kun deler av et kjøretøy vises. Begge disse forbedringene vil kreve et større datasett med en jevnere representasjon av alle inkluderte klasser.

Målet med dette studiet var at deteksjonen av kjøretøy skal bli god nok til at en oppnå pålitelig sporing av kjøretøy. En viktig problemstilling knyttet til dette, og som ikke har blitt sett på her, er algoritmens evne til å gjennomføre deteksjon og sporing av kjøretøy på livevideo i sanntid. I tillegg til at algoritmen må kunne gjøre dette på en lettvekt maskinvare som for eksempel Coral USB Accelerator. Dette vil være essensielt for at modellen skal kunne konkurrere med dagens løsning.

I dette prosjektet er DeepSORT blitt testet som sporingsalgoritme, og av den grunn hadde det vært interessant å utforske dette ytterligere i videre arbeid.

8 Broader Impacts

Denne oppgaven anvender eksisterende teknologi og bruker den til deteksjon av biler. Målet var å forbedre presisjon og ikke endre funksjonaliteten til Q-Frees nåværende systemer. De momentane og fremtidige etiske konsekvensene for samfunnet som dette arbeidet kan medbringe er derfor begrenset til de generelle konsekvensene av økt bruk av maskinlæring. I dette kapittelet drøftes derfor *Broader Impact* for videreutvikling og anvendelse av CV-arbeid utover prosjektets direkte innvirkning.

Videre utvikling og forbedring av området innenfor maskinlæring som omhandler objektdeteksjon og objektsporing, kan bidra til å automatisere dagens systemer. Dette vil innebære å ta stilling til bruksområdet og innvirkningen arbeidet kan resultere i. Objektdeteksjon er et område som i enkelte tilfeller kan misbrukes til å krenke personvern, eksempelvis gjennom militær overvåking av mennesker. Joseph Redmon, utvikleren bak de tre første YOLO-versjonene, valgte på dette etiske grunnlaget å avslutte alt CV-arbeid han i utgangspunktet trives med [68]. På et spørsmål om når man skal la være å publisere en forskningsartikkel, på grunnlag av påvirkningen den kan ha i fremtiden, svarer Redmon at han måtte avslutte arbeidet på grunn av personvernhensyn og at han ikke lenger kunne overse det militære bruksområdet. Selv om maskinlæring er et verktøy som har stort potensial til å forbedre flere av dagens systemer, er det viktig å ta stilling til etiske problemstillinger ved bruk av teknologi som kan misbrukes.

Bruken av maskinlæring har økt de siste årene, og videreutviklingen av dette gjør dem mer kjent og anvendt i markedet. Studier som dette prosjektet gir en redegjørelse for hva som kan oppnås med dagens algoritmer, samt begrensningene de har. Datasettet som er benyttet består av privatbiler, og det er derfor viktig å ta forholdsregler for å bevare personvern. Løsningen som er kommet fram til i dette studiet kan være et supplement til dagens løsning for å øke presisjon. Den tilbyr ingen ny funksjonalitet til systemet. Det vil derfor ikke være mulig å identifisere mennesker eller bilskilt slik løsningen er i dag.

9 Referanser

[1] Coral USB Accelerator. https://coral.ai/products/accelerator/. [Hentet mai 2021].

[2] Andreas Kaplan and Michael Haenlein. Siri, siri, in my hand: Who's the fairest in the land? on the interpretations, illustrations, and implications of artificial intelligence. *Business Horizons*, 62(1):15–25, jan 2019. doi: 10.1016/j.bushor.2018.08.004. URL https://doi.org/10.1016%2Fj.bushor.2018.08.004.

[3] Michael Haenlein and Andreas Kaplan. A brief history of artificial intelligence: On the past, present, and future of artificial intelligence. *California Management Review*, 61(4):5–14, 2019. doi: 10.1177/0008125619864925. URL https://doi.org/10.1177/0008125619864925.

[4] IBM Cloud Education. Supervised Learning. https://www.ibm.com/cloud/learn/supervised-learning, 2020. [Hentet mar. 2021].

[5] Devin Soni. Supervised vs. Unsupervised Learning. https://towardsdatascience.com/supervised-vs-unsupervised-learning-14f68e32ea8d, 2018. [Hentet mai 2021].

[6] Avinash Navlani. Neural Network Model in R. https://www.datacamp.com/community/tutorials/neural-network-models-r, 2019. [Hentet feb. 2021].

[7] Abhijit Roy. An Introduction to Gradient Descent and Backpropagation. https://towardsdatascience.com/an-introduction-to-gradient-descent-and-backpropagation-81648bdb19b2, 2020. [Hentet apr. 2021].

[8] Xue Ying. An overview of overfitting and its solutions. https://iopscience.iop.org/article/10.1088/1742-6596/1168/2/022022/pdf, 2019. [Hentet mai 2021].

[9] Steve Lawrence, C Lee Giles, and Ah Chung Tsoi. Lessons in neural network training: Overfitting may be harder than expected. https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.48.8002&rep=rep1&type=pdf, 1997. [Hentet mai 2021].

[10] Umme Zahoora Aqsa Saeed Qureshi Asifullah Khan, Anabia Sohail. A survey of the recent architectures of deep convolutional neural networks. https://link.springer.com/article/10.1007/s10462-020-09825-6#citeas, 2020. [Hentet apr. 2021].

[11] Vojtech Pavlovsky. Introduction To Convolutional Neural Networks. https://www.vojtech.net/posts/intro-convolutional-neural-networks/, 2017. [Hentet apr. 2021].

[12] Afshine Amidi and Shervine Amidi. Convolutional neural networks cheatsheet. https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-convolutional-neural-networks. [Hentet apr. 2021].

[13] Joseph Redmon. Darknet: Open Source Neural Networks in C. http://pjreddie.com/darknet/, 2013–2016. [Hentet apr. 2021].

[14] Licheng Jiao, Fan Zhang, Fang Liu, Shuyuan Yang, Lingling Li, Zhixi Feng, and Rong Qu. A Survey of Deep Learning-based Object Detection. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8825470, 2019. [Hentet feb. 2021].

[15] Xiongwei Wu, Doyen Sahoo, and Steven CH Hoi. Recent Advances in Deep Learning for Object Detection. https://arxiv.org/pdf/1908.03673.pdf, 2020. [Hentet feb. 2021].

[16] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. YOLOv4: Optimal Speed and Accuracy of Object Detection. https://arxiv.org/abs/2004.10934, 2020. [Hentet jan. 2021].

[17] Huaiyu Li Zhe Dai Xu Yun Huansheng Song, Haoxiang Liang. Vision-based vehicle detection and counting system using deep learning in highway scenes. https://etrr.springeropen.com/articles/10.1186/s12544-019-0390-4, 2019. [Hentet mai 2021].

[18] Shivy Yohanandan. map (mean average precision) might confuse you. https://towardsdatascience.com/map-mean-average-precision-might-confuse-you-5956f1bfa9e2, 2019. [Hentet apr. 2021].

[19] Rafael Padilla, Sergio Netto, and Eduardo da Silva. A Survey on Performance Metrics for Object-Detection Algorithms. https://www.researchgate.net/publication/343194514_A_Survey_on_Performance_Metrics_for_Object-Detection_Algorithms, 2020. [Hentet mai 2021].

[20] Thomas Wood. What is the f-score? https://deepai.org/machine-learning-glossary-and-terms/f-score. [Hentet mai 2021].

[21] Jonathan Hui. map (mean average precision) for object detection. https://jonathan-hui.medium.com/map-mean-average-precision-for-object-detection-45c121a31173, 2018. [Hentet mai 2021].

[22] Yilmaz, Alper and Javed, Omar and Shah, Mubarak. Object tracking: A survey. https://www.researchgate.net/profile/Alper-Yilmaz-9/publication/220566062_Object_tracking_a_survey_ACM_Comput_Surv/links/

0c960528a302b64f59000000/Object-tracking-a-survey-ACM-Comput-Surv.pdf, 2006. [Hentet feb. 2021].

[23] N. Wojke, A. Bewley, and D. Paulus. Simple Online and Realtime Tracking with a Deep Association Metric. https://arxiv.org/abs/1703.07402, 2017. [Hentet jan. 2021].

[24] Object Tracking in Deep Learning. https://missinglink.ai/guides/computer-vision/object-tracking-deep-learning/. [Hentet feb. 2021].

[25] Weiqiang Li, Jiatong Mu, and Guizhong Liu. Multiple Object Tracking with Motion and Appearance Cues. https://arxiv.org/ftp/arxiv/papers/1909/1909.00318.pdf, 2019. [Hentet mar. 2021].

[26] Gioele Ciaparrone, Francisco Luque Sánchez, Siham Tabik, Luigi Troiano, Roberto Tagliaferri, and Francisco Herrera. Deep Learning in Video Multi-Object Tracking: A Survey. https://arxiv.org/pdf/1907.12740.pdf, 2020. [Hentet mar. 2021].

[27] Li, Chenge and Dobler, Gregory and Feng, Xin and Wang, Yao. TrackNet: Simultaneous Object Detection and Tracking and Its Application in Traffic Video Analysis. https://arxiv.org/pdf/1902.01466.pdf, 2019. [Hentet mar. 2021].

[28] Anthony D. Rhodes. A Overview of Computer Vision Tasks, including Multiple-Object Detection (MOT). http://web.pdx.edu/~arhodes/CV_MOT.pdf, 2018. [Hentet mar. 2021].

[29] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. You Only Look Once: Unified, Real-Time Object Detection. http://arxiv.org/abs/1506.02640, 2015. [Hentet jan. 2021].

[30] Karlijn Alderliesten. YOLOv3 - Real-time object detection. https://medium.com/analytics-vidhya/yolov3-real-time-object-detection-54e69037b6d0. [Hentet apr. 2021].

[31] Zicong Jiang, Liquan Zhao, Shuaiyang Li, and Yanfei Jia. Real-time object detection method based on improved YOLOv4-tiny. https://arxiv.org/abs/2011.04244, 2020. [Hentet feb. 2021].

[32] Luis Silva, Héctor Sánchez San Blas, David García, André Mendes, and G. Villarrubia. An Architectural Multi-Agent System for a Pavement Monitoring System with Pothole Recognition in UAV Images. https://www.researchgate.net/publication/345212328_An_Architectural_Multi-Agent_System_for_a_Pavement_Monitoring_System_with_Pothole_Recognition_in_UAV_Images, 2020. [Hentet apr. 2021].

[33] Deepaloke Chattopadhyay, Sania Rasheed, Luyuanyuan Yan, Alfonso A. Lopez, Jay Farmer, and Donald E. Brown. Machine Learning for Real-Time Vehicle

Detection in All-Electronic Tolling System. https://ieeexplore.ieee.org/abstract/document/9106682, 2020. [Hentet mai 2021].

[34] Yundong LI, Han DONG, Hongguang LI, Xueyan ZHANG, Baochang ZHANG, and Zhifeng XIAO. Multi-block SSD based on small object detection for UAV railway scene surveillance. *Chinese journal of aeronautics*, 33(6):1747–1755, 2020. ISSN 1000-9361. [Hentet apr. 2021].

[35] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. SSD: Single Shot MultiBox Detector. In *Computer Vision – ECCV 2016*, Lecture Notes in Computer Science, pages 21–37, Cham, 2016. Springer International Publishing. ISBN 3319464477.

[36] Ruhin Mary Saji and N V Sobhana. Real time object detection using ssd for bank security. *IOP conference series. Materials Science and Engineering*, 1070 (1):12060, 2021. ISSN 1757-8981. [Hentet mar. 2021].

[37] George Seif. Visualising filters and feature maps for deep learning. https://towardsdatascience.com/visualising-filters-and-feature-maps-for-deep-learning-d814e13bd671, 2019. [Hentet apr. 2021].

[38] Baoqing Guo, Jiafeng Shi, Liqiang Zhu, and Zujun Yu. High-speed railway clearance intrusion detection with improved ssd network. *Applied sciences*, 9(15):2981, 2019. ISSN 2076-3417.

[39] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. 2017.

[40] MATLAB. Understanding Kalman Filters, Part 1: Why Use Kalman Filters? https://www.youtube.com/watch?v=mwn8xhgNpFY, 2017. [Hentet feb. 2021].

[41] H. W. Kuhn. The hungarian method for the assignment problem. https://link.springer.com/content/pdf/10.1007%2F978-3-540-68279-0_2.pdf, 1955. [Hentet mar. 2021].

[42] Zhengjun Qiu, Nan Zhao, Lei Zhou, Mengcen Wang, Liangliang Yang, Hui Fang, Yong He, and Yufei Liu. Vision-based moving obstacle detection and tracking in paddy field using improved yolov3 and deep sort. https://www.researchgate.net/publication/343172917_Vision-Based_Moving_Obstacle_Detection_and_Tracking_in_Paddy_Field_Using_Improved_Yolov3_and_Deep_SORT, 2020. [Hentet apr. 2021].

[43] Anushka Dhiman. Object Tracking using DeepSORT in TensorFlow 2. https://medium.com/analytics-vidhya/

object-tracking-using-deepsort-in-tensorflow-2-ec013a2eeb4f, 2020. [Hentet mai 2021].

[44] Vishal Mandal and Yaw Adu-Gyamfi. Object Detection and Tracking Algorithms for Vehicle Counting: A Comparative Analysis. https://arxiv.org/abs/2007.16198, 2020. [Hentet jan. 2021].

[45] Tzutalin. Labelimg. https://github.com/tzutalin/labelImg]. [Hentet mai 2021].

[46] Chanjoo Lee (ZZANZU). Yolo-convert-txt-2-xml. https://github.com/ZZANZU/YOLO-convert-txt-2-xml, 2020. [Hentet mai. 2021].

[47] The TensorFlow Authors. Tfrecord and tf.train.example. https://www.tensorflow.org/tutorials/load_data/tfrecord, 2021. [Hentet apr. 2021].

[48] Khush Patel. xml_to_csv.py. https://gist.github.com/iKhushPatel/ed1f837656b155d9b94d45b42e00f5e4. [Hentet feb. 2021].

[49] Dat Tran. generate_tfrecord.py. https://github.com/datitran/raccoon_dataset/blob/master/generate_tfrecord.py. [Hentet mar. 2021].

[50] The TensorFlow Authors. Tensorflow object detection api. https://github.com/tensorflow/models/tree/master/research/object_detection, 2021. [Hentet feb. 2021].

[51] TensorFlow Authors. TensorFlow 2 model zoo. https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf2_detection_zoo.md. [Hentet mar. 2021].

[52] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft coco: Common objects in context. In *Computer Vision – ECCV 2014*, Lecture Notes in Computer Science, pages 740–755. Springer International Publishing, Cham. ISBN 3319106015.

[53] AlexeyAB. Yolo v4, v3 and v2 for Windows and Linux. https://github.com/AlexeyAB/darknet. [Hentet feb. 2021].

[54] Odemakinde Elisha Jesutofunmi. real_time.py. https://github.com/elishatofunmi/Computer-Vision/blob/master/static%20video%20object%20detection%20and%20tracking/real_time.py, 2020. [Hentet mar. 2021].

[55] Santiago Valdarrama. confusion_matrix_tf2.py. https://github.com/svpino/tf_object_detection_cm/blob/master/confusion_matrix_tf2.py, 2020. [Hentet mar. 2021].

[56] hunglc007. -yolov4-tflite. https://github.com/hunglc007/tensorflow-yolov4-tflite. [Hentet mar. 2021].

[57] theAIGuysCode. yolov4-custom-functions. https://github.com/theAIGuysCode/yolov4-custom-functions, . [Hentet mar. 2021].

[58] theAIGuysCode. yolov4-deepsort. https://github.com/theAIGuysCode/yolov4-deepsort, . [Hentet mai 2021].

[59] Tensorflow. Tensorflow Core. https://www.tensorflow.org/overview. [Hentet feb. 2021].

[60] OpenCV - About. https://opencv.org/about/. [Hentet mai 2021].

[61] opencv. Opencv: Open source computer vision library. https://github.com/opencv/opencv, . [Hentet mai 2021].

[62] opencv. Repository for OpenCV's extra modules. https://github.com/opencv/opencv_contrib, . [Hentet mai 2021].

[63] CUDA. https://developer.nvidia.com/cuda-zone, . [Hentet mai 2021].

[64] NVIDIA cuDNN. https://developer.nvidia.com/cudnn, . [Hentet mar. 2021].

[65] Idun. Idun. https://www.hpc.ntnu.no/idun. [Hentet apr. 2021].

[66] Magnus Sjalander, Magnus Jahre, Gunnar Tufte, and Nico Reissmann. EPIC: An Energy-Efficient, High-Performance GPGPU Computing Research Infrastructure. https://arxiv.org/pdf/1912.05848.pdf, 2019. [Hentet apr. 2021].

[67] Andy B. Yoo, Morris A. Jette, and Mark Grondona. Slurm: Simple linux utility for resource management. In Dror Feitelson, Larry Rudolph, and Uwe Schwiegelshohn, editors, *Job Scheduling Strategies for Parallel Processing*, pages 44–60, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg. ISBN 978-3-540-39727-4.

[68] Joseph Redmon. "this is huge. the creator of the yolo algorithms, which (along with ssd) set much of the path of modern object detection, has stopped doing any computer vision research due to ethical concerns. i've never seen anything quite like this before.". https://twitter.com/pjreddie/status/1230524770350817280. [Hentet mai 2021].

10 Vedlegg

A Original oppgavetekst

Arbeidstittel:	Maskinlæringsoppgave - Q-Free ASA
Hensikten med oppgaven: Å bruke maskinlæring og Googles nye Coral-chip (https://coral.ai/) til å prøve å detektere gjenstander på biler eller også hele biler gjennom sonen under en bomstasjon.	
Kort beskrivelse av oppgaveforslag: Systemgruppa i Q-Free jobber med leveranser av programvare til Q-Frees bomstasjoner rundt om i verden. En sentral del av dette arbeidet dreier seg rundt kameraprogramvare. I Q-Frees gjeldende generasjon utstyr bruker vi vidvinkelkameraer til å spore bilder gjennom bomstasjonen for å koble sammen bak- og frontskilt på biler. Tradisjonelt sett har vi ikke brukt maskinlæring til dette men dette er noe vi nå ønsker å utforske. Oppgaven går ut på å bruke maskinlæring og Googles nye Coral-chip (https://coral.ai/) til å prøve å detektere gjenstander på biler eller også hele biler gjennom sonen under en bomstasjon. Vi stiller med datasett og oppgaven går ut på å trene opp TensorFlow-nettverk samt å bruke disse til å kvantifisere, dvs. statistisk stadfeste i hvor stor grad de virker når det kommer til sporing av f.eks. skilt, frontruter, lykter, hele biler osv. Oppgaven kan godt løses i Python som et prototypingsprosjekt. Om student(ene) ønsker det kan oppgaven også innebære å porte prototype-kode over i C++. Vi har allerede et eksisterende rammeverk rundt dette og vi bidrar med veiledning underveis.	
Oppgaven passer for (kryss av de(t) som passer og skriv evt. en kommentar til oss):	- Bacheloroppgave - Masteroppgave - Prosjektoppgave, Dataingeniør (Systemutvikling og/eller 3D-grafikk)
Kan oppgavestiller stille arbeidsplass med nødvendig utstyr og programvare:	Ja
Begrensninger i tilgjenglighet av opplysninger o.l:	NDA kan være nødvendig. Må diskuteres nærmere.
Oppgaven passer best for, antall studenter:	- 1 - 2 - 3
Opplysninger om oppgavestiller	
Er du fra bedrift/virksomhet eller er du student med en egendefinert/selvlaget oppgave?	- Bedrift/virksomhet
Navn på bedrift/organisasjon/student:	Q-Free ASA
Addresse	Strindfjordveien 1
Postnummer	7053
Poststed	Ranheim
Navn på kontaktperson/veileder:	Torstein Malvik
Telefon:	95901092
Epost:	torstein.malvik@q-free.com
Navn på kontaktperson 2/veileder 2:	Karianne Sæter
Telefon kontaktperson 2/veileder 2:	48198090
Epost kontaktperson 2/veileder 2:	karianne.saeter@q-free.com

B Videoer

Videoer med deteksjoner: Link til deteksjonsvideoer

Videoer med sporing: Link til sporingsvideoer

C config SSD_mobilenet_V1_UT

```
model {
  ssd {
    num_classes: 1
    image_resizer {
      fixed_shape_resizer {
        height: 640
        width: 640
      }
    }
    feature_extractor {
      type: "ssd_mobilenet_v1_fpn_keras"
      depth_multiplier: 1.0
      min_depth: 16
      conv_hyperparams {
        regularizer {
          l2_regularizer {
            weight: 4e-05
          }
        }
        initializer {
          random_normal_initializer {
            mean: 0.0
            stddev: 0.01
          }
        }
        activation: RELU_6
        batch_norm {
          decay: 0.997
          scale: true
          epsilon: 0.001
        }
      }
      override_base_feature_extractor_hyperparams: true
      fpn {
        min_level: 3
        max_level: 7
      }
    }
    box_coder {
```

```
    faster_rcnn_box_coder {
      y_scale: 10.0
      x_scale: 10.0
      height_scale: 5.0
      width_scale: 5.0
    }
  }
  matcher {
    argmax_matcher {
      matched_threshold: 0.5
      unmatched_threshold: 0.5
      ignore_thresholds: false
      negatives_lower_than_unmatched: true
      force_match_for_each_row: true
      use_matmul_gather: true
    }
  }
  similarity_calculator {
    iou_similarity {
    }
  }
  box_predictor {
    weight_shared_convolutional_box_predictor {
      conv_hyperparams {
        regularizer {
          l2_regularizer {
            weight: 4e-05
          }
        }
        initializer {
          random_normal_initializer {
            mean: 0.0
            stddev: 0.01
          }
        }
        activation: RELU_6
        batch_norm {
          decay: 0.997
          scale: true
          epsilon: 0.001
        }
```

```
      }
      depth: 256
      num_layers_before_predictor: 4
      kernel_size: 3
      class_prediction_bias_init: -4.6
    }
  }
  anchor_generator {
    multiscale_anchor_generator {
      min_level: 3
      max_level: 7
      anchor_scale: 4.0
      aspect_ratios: 1.0
      aspect_ratios: 2.0
      aspect_ratios: 0.5
      scales_per_octave: 2
    }
  }
  post_processing {
    batch_non_max_suppression {
      score_threshold: 1e-08
      iou_threshold: 0.6
      max_detections_per_class: 100
      max_total_detections: 100
      use_static_shapes: false
    }
    score_converter: SIGMOID
  }
  normalize_loss_by_num_matches: true
  loss {
    localization_loss {
      weighted_smooth_l1 {
      }
    }
    classification_loss {
      weighted_sigmoid_focal {
        gamma: 2.0
        alpha: 0.25
      }
    }
    classification_weight: 1.0
```

```
      localization_weight: 1.0
    }
    encode_background_as_zeros: true
    normalize_loc_loss_by_codesize: true
    inplace_batchnorm_update: true
    freeze_batchnorm: false
  }
}
train\_config {
  batch_size: 16
  data_augmentation_options {
    random_horizontal_flip {
    }
  }
  data_augmentation_options {
    random_crop_image {
      min_object_covered: 0.0
      min_aspect_ratio: 0.75
      max_aspect_ratio: 3.0
      min_area: 0.75
      max_area: 1.0
      overlap_thresh: 0.0
    }
  }
  sync_replicas: true
  optimizer {
    momentum_optimizer {
      learning_rate {
        cosine_decay_learning_rate {
          learning_rate_base: 0.04
          total_steps: 6000
          warmup_learning_rate: 0.013333
          warmup_steps: 500
        }
      }
      momentum_optimizer_value: 0.9
    }
    use_moving_average: false
  }
  fine_tune_checkpoint: "pre-trained-models/
     ssd_mobilenet_v1_fpn_640x640_coco17_tpu-8/checkpoint/ckpt
```

```
      -0"
  num_steps: 6000
  startup_delay_steps: 0.0
  replicas_to_aggregate: 8
  max_number_of_boxes: 100
  unpad_groundtruth_tensors: false
  fine_tune_checkpoint_type: "detection"
  fine_tune_checkpoint_version: V2
}
train\_input\_reader {
  tf_record_input_reader {
    input_path: "annotations/Records_04/train.record"
  }
}
eval\_config {
  metrics_set: "coco_detection_metrics"
  use_moving_averages: false
  batch_size: 1
}
eval\_input\_reader {
  label_map_path: "annotations/Records_04/label_map.pbtxt"
  shuffle: false
  num_epochs: 1
  tf_record_input_reader {
    input_path: "annotations/Records_04/test.record"
  }
}
```

D config SSD_mobilenet_V2_UT

```
model {
  ssd {
    num_classes: 1
    image_resizer {
      fixed_shape_resizer {
        height: 320
        width: 320
      }
    }
    feature_extractor {
      type: "ssd_mobilenet_v2_keras"
      depth_multiplier: 1.0
      min_depth: 16
      conv_hyperparams {
        regularizer {
          l2_regularizer {
            weight: 4e-05
          }
        }
        initializer {
          truncated_normal_initializer {
            mean: 0.0
            stddev: 0.03
          }
        }
        activation: RELU_6
        batch_norm {
          decay: 0.97
          center: true
          scale: true
          epsilon: 0.001
          train: true
        }
      }
      override_base_feature_extractor_hyperparams: true
    }
    box_coder {
      faster_rcnn_box_coder {
        y_scale: 10.0
```

```
      x_scale: 10.0
      height_scale: 5.0
      width_scale: 5.0
    }
  }
  matcher {
    argmax_matcher {
      matched_threshold: 0.5
      unmatched_threshold: 0.5
      ignore_thresholds: false
      negatives_lower_than_unmatched: true
      force_match_for_each_row: true
      use_matmul_gather: true
    }
  }
  similarity_calculator {
    iou_similarity {
    }
  }
  box_predictor {
    convolutional_box_predictor {
      conv_hyperparams {
        regularizer {
          l2_regularizer {
            weight: 4e-05
          }
        }
        initializer {
          random_normal_initializer {
            mean: 0.0
            stddev: 0.01
          }
        }
        activation: RELU_6
        batch_norm {
          decay: 0.97
          center: true
          scale: true
          epsilon: 0.001
          train: true
        }
```

```
      }
      min_depth: 0
      max_depth: 0
      num_layers_before_predictor: 0
      use_dropout: false
      dropout_keep_probability: 0.8
      kernel_size: 1
      box_code_size: 4
      apply_sigmoid_to_scores: false
      class_prediction_bias_init: -4.6
    }
  }
  anchor_generator {
    ssd_anchor_generator {
      num_layers: 6
      min_scale: 0.2
      max_scale: 0.95
      aspect_ratios: 1.0
      aspect_ratios: 2.0
      aspect_ratios: 0.5
      aspect_ratios: 3.0
      aspect_ratios: 0.3333
    }
  }
  post_processing {
    batch_non_max_suppression {
      score_threshold: 1e-08
      iou_threshold: 0.6
      max_detections_per_class: 100
      max_total_detections: 100
      use_static_shapes: false
    }
    score_converter: SIGMOID
  }
  normalize_loss_by_num_matches: true
  loss {
    localization_loss {
      weighted_smooth_l1 {
        delta: 1.0
      }
    }
```

```
      classification_loss {
        weighted_sigmoid_focal {
          gamma: 2.0
          alpha: 0.75
        }
      }
      classification_weight: 1.0
      localization_weight: 1.0
    }
    encode_background_as_zeros: true
    normalize_loc_loss_by_codesize: true
    inplace_batchnorm_update: true
    freeze_batchnorm: false
  }
}
train_config {
  batch_size: 16
  data_augmentation_options {
    random_horizontal_flip {
    }
  }
  data_augmentation_options {
    ssd_random_crop {
    }
  }
  sync_replicas: true
  optimizer {
    momentum_optimizer {
      learning_rate {
        cosine_decay_learning_rate {
          learning_rate_base: 0.08000001
          total_steps: 6000
          warmup_learning_rate: 0.013333002
          warmup_steps: 500
        }
      }
      momentum_optimizer_value: 0.9
    }
    use_moving_average: false
  }
  fine_tune_checkpoint: "pre-trained-models/ssd_mobilenet_v2_320x320_coco17
```

```
  num_steps: 6000
  startup_delay_steps: 0.0
  replicas_to_aggregate: 8
  max_number_of_boxes: 100
  unpad_groundtruth_tensors: false
  fine_tune_checkpoint_type: "detection"
  fine_tune_checkpoint_version: V2
}
train_input_reader {
  label_map_path: "annotations/Records_04/label_map.pbtxt"
  tf_record_input_reader {
    input_path: "annotations/Records_04/train.record"
  }
}
eval_config {
  metrics_set: "coco_detection_metrics"
  use_moving_averages: false
}
eval_input_reader {
  label_map_path: "annotations/Records_04/label_map.pbtxt"
  shuffle: false
  num_epochs: 1
  tf_record_input_reader {
    input_path: "annotations/Records_04/test.record"
  }
}
```

E config SSD_mobilenet_V1_MT

```
model {
  ssd {
    num_classes: 1
    image_resizer {
      fixed_shape_resizer {
        height: 640
        width: 640
      }
    }
    feature_extractor {
      type: "ssd_mobilenet_v1_fpn_keras"
      depth_multiplier: 1.0
      min_depth: 16
      conv_hyperparams {
        regularizer {
          l2_regularizer {
            weight: 4e-05
          }
        }
        initializer {
          random_normal_initializer {
            mean: 0.0
            stddev: 0.01
          }
        }
        activation: RELU_6
        batch_norm {
          decay: 0.997
          scale: true
          epsilon: 0.001
        }
      }
      override_base_feature_extractor_hyperparams: true
      fpn {
        min_level: 3
        max_level: 7
      }
    }
    box_coder {
```

```
    faster_rcnn_box_coder {
      y_scale: 10.0
      x_scale: 10.0
      height_scale: 5.0
      width_scale: 5.0
    }
  }
  matcher {
    argmax_matcher {
      matched_threshold: 0.5
      unmatched_threshold: 0.5
      ignore_thresholds: false
      negatives_lower_than_unmatched: true
      force_match_for_each_row: true
      use_matmul_gather: true
    }
  }
  similarity_calculator {
    iou_similarity {
    }
  }
  box_predictor {
    weight_shared_convolutional_box_predictor {
      conv_hyperparams {
        regularizer {
          l2_regularizer {
            weight: 4e-05
          }
        }
        initializer {
          random_normal_initializer {
            mean: 0.0
            stddev: 0.01
          }
        }
        activation: RELU_6
        batch_norm {
          decay: 0.997
          scale: true
          epsilon: 0.001
        }
```

```
      }
      depth: 256
      num_layers_before_predictor: 4
      kernel_size: 3
      class_prediction_bias_init: -4.6
    }
  }
  anchor_generator {
    multiscale_anchor_generator {
      min_level: 3
      max_level: 7
      anchor_scale: 4.0
      aspect_ratios: 1.0
      aspect_ratios: 2.0
      aspect_ratios: 0.5
      scales_per_octave: 2
    }
  }
  post_processing {
    batch_non_max_suppression {
      score_threshold: 1e-08
      iou_threshold: 0.6
      max_detections_per_class: 100
      max_total_detections: 100
      use_static_shapes: false
    }
    score_converter: SIGMOID
  }
  normalize_loss_by_num_matches: true
  loss {
    localization_loss {
      weighted_smooth_l1 {
      }
    }
    classification_loss {
      weighted_sigmoid_focal {
        gamma: 2.0
        alpha: 0.25
      }
    }
    classification_weight: 1.0
```

```
      localization_weight: 1.0
    }
    encode_background_as_zeros: true
    normalize_loc_loss_by_codesize: true
    inplace_batchnorm_update: true
    freeze_batchnorm: false
  }
}
train_config {
  batch_size: 8
  data_augmentation_options {
    random_horizontal_flip {
    }
  }
  data_augmentation_options {
    random_crop_image {
      min_object_covered: 0.0
      min_aspect_ratio: 0.75
      max_aspect_ratio: 3.0
      min_area: 0.75
      max_area: 1.0
      overlap_thresh: 0.0
    }
  }
  sync_replicas: true
  optimizer {
    momentum_optimizer {
      learning_rate {
        cosine_decay_learning_rate {
          learning_rate_base: 0.089999996
          total_steps: 16000
          warmup_learning_rate: 0.013333
          warmup_steps: 500
        }
      }
      momentum_optimizer_value: 0.9
    }
    use_moving_average: false
  }
  fine_tune_checkpoint: "pre-trained-models/ssd_mobilenet_v1_fpn_640x640_co
  num_steps: 16000
```

```
  startup_delay_steps: 0.0
  replicas_to_aggregate: 8
  max_number_of_boxes: 100
  unpad_groundtruth_tensors: false
  fine_tune_checkpoint_type: "detection"
  fine_tune_checkpoint_version: V2
}
train_input_reader {
  tf_record_input_reader {
    input_path: "annotations/Records_03/train.record"
  }
}
eval_config {
  metrics_set: "coco_detection_metrics"
  use_moving_averages: false
  batch_size: 1
}
eval_input_reader {
  label_map_path: "annotations/Records_03/label_map.pbtxt"
  shuffle: false
  num_epochs: 1
  tf_record_input_reader {
    input_path: "annotations/Records_03/test.record"
  }
}
```