Høgskulen på Vestlandet

BACHELOROPPGAVE

Detektering og varsling av hjortevilt ved veibane ved bruk av dyp læring

Detection and warning of game in vicinity of roads with deep learning

Thor Alme Aasheim

Joachim William Eriksen Leiros

Andreas Nordang Seljeset

Fakultet for ingeniør- og naturvitskap

Institutt for datateknologi, elektroteknologi og realfag

Informasjonsteknologi

Veileder: Per Christian Engdal

Innleveringsdato: 23/05/2022

Høgskulen på Vestlandet

Fakultet for ingeniør- og naturvitskap

Institutt for datateknologi, elektroteknologi og realfag

TITTELSIDE FOR HOVEDPROSJEKT

Rapportens tittel:	*Dato:*
Detektering og varsling av hjort ved veibane ved bruk av dyp læring	*27/02-2022*
Forfatter(e):	*Antall sider u/vedlegg: 62*
Thor Alme Aasheim, Joachim William Eriksen Leiros, Andreas Nordang Seljeset	*Antall sider vedlegg: 121*
Studieretning:	*Antall disketter/CD-er:*
Informasjonsteknologi	*0*
Kontaktperson ved studieretning:	*Gradering:*
Per Christian Engdal	Ingen
Merknader:	
Ingen	

Oppdragsgiver:	*Oppdragsgivers referanse:*
HVL-IDER	IF-1
Oppdragsgivers kontaktperson:	*Telefon:*
Martin Stølen	+47 57 72 25 03

Sammendrag:

Målet med prosjektet er å lage en maskinlæringsmodell som kan detektere hjortevilt basert på inndata fra et kamera som sendte bilder i farger, dersom hjortevilt ble oppdaget skal systemet sende ut et varsel. Ved bruk av YOLOv5 og Python ble det utviklet et programsystem som tar inndata fra en URL eller kamera, bildet blir prosessert og det blir predikert om det er hjort i bildet. Ved deteksjon sendes det et varsel. YOLOv5-Medium modellen hadde som resultat mAP@0.945 og 1492ms i prosesseringstid per bilde, og YOLOv5-Small mAP@0.898 med 627ms per bilde.

The goal is to create a machine learning model which can detect game based on input from a camera which captures images in RGB-colors. If game is detected the system will send out a warning. By using YOLOv5 and Python, a program which takes video input from an URL or a camera was developed. The image is processed and the model predicts if there is deer in the image. If there is a detection, a warning is sent. The YOLOv5-Medium model had a result of mAP@0.945 and 1492ms processing time per image, YOLOv5-Small had mAP@0.898 with 627ms per image.

Stikkord:

Maskinlæring	Objekt detektering	Påkjørsel av vilt

Forord

Denne rapporten dokumenterer arbeidet gjort på bachelorprosjektet "*Detektering og varsling av hjortevilt ved veibane ved bruk av dyp læring", ved Høgskulen på Vestlandet (HVL). Prosjektet ble utført av Thor Aasheim, Joachim Leiros og Andreas Seljeset våren 2022.*

Vi ønsker å takke:

- Per Christian Engdal for god veiledning på rapport, systemutvikling og det administrative i løpet av prosjektet.
- Martin Fodstad Stølen for muligheten til å gjennomføre dette spennende prosjektet.
- Stein Joar Hegland for muligheten til å gjennomføre dette prosjektet, bidrag med anskaffelse av bildemateriale samt ekspertise om hjortevilt.
- Alf Erik Røyrvik for bidrag med anskaffelse av bildemateriale.
- Krister Seljeset for bidrag med anskaffelse av bildemateriale.
- Reza Arghandeh for hjelp med analyse og valg av algoritme.
- Truls Erlend Jørgensen for ideen bak prosjektet.

INNHOLDSFORTEGNELSE

1 INNLEDNING

I dette kapittelet gis en innføring i bakgrunnen til problemet og forklarer videre hvorfor det må løses. Deretter introduseres prosjektets mål og eier.

1.1 Kontekst

Dagens system for varsling av hjortevilt baserer seg på en beregning av sannsynligheten for at det er vilt i området på et visst tidspunkt *(Dynamiske hjorteskilt i Vestland fylke, 2017-2021).* Da varsling ikke er basert på reell detektering medfører dette høy forekomst av falske- positive og negative varslinger. Dette resulterer i at trafikanter har liten tiltro til varslingen.

Teknologi for objekt deteksjon ved bruk av kamera og maskinlæringsalgoritmer har i de siste år vært i stor utvikling (Russakovsky, 2015). Høgskulen på Vestlandet(HVL) sin gruppe for Data og Automasjon med robotikk ønsker å undersøke hvordan denne teknologien kan utnyttes for å øke treffsikkerheten på varslingen og redusere antall påkjørsler av hjortevilt.

1.2 Motivasjon

I 2019 gjennomførte Statistisk Sentralbyrå en analyse (*Skara, 2019*) over viltpåkjørsler i Norge. Studiet viser at rundt 8 800 hjortevilt drepes i trafikken hvert år. Dette fører til store utgifter for trafikanter, forsikringsselskap, fylkeskommune, kommune og Statens Vegvesen.

De årlige samfunnskostnadene som resultat av påkjørsel av hjortevilt er estimert til å være på mellom 800 millioner og 1 milliard kroner årlig. (IMSA Knowledge Company AS, vedlegg med etterord fra IMSA i rapporten til Hegland og Frøyen (*Hegland og Frøyen, 2021)*.

Det er varslingssystem for hjortevilt i aktiv bruk i dag, men varslingen er for unøyaktig. Etter å ha gjennomført et forsøk med bruk av dynamiske hjorteskilt i Vestland fylke 2017-2021 viser resultatene at varslingen ofte gir falsk- positiv eller negativ varsling. Varslingen baserer seg på en sannsynlighetsverdi som blir beregnet utfra tid, sted og sesong (*Hegland og Frøyen, 2021)*.

Høy frekvens av feilvarsling har resultert i at trafikanter har liten tiltro til varslingen som medfører liten eller ingen fartsreduksjon selv ved varsling av høy fare for hjortevilt. Ved å gå over til varsling ved direkte deteksjon vil sannsynligheten for falsk positiv og negativ varsling reduseres. Dette kan forbedre trafikanters tiltro til systemet hvor de vil redusere fart ved varsling. Lavere hastighet vil føre til færre kollisjoner, redusert skadeomfang og reduserte kostnader forbundet med person- og materialskader for trafikanter, kommune, fylkeskommune, Statens vegvesen og forsikringsselskap.

1.3 Prosjekteier

Prosjekteier er HVL Institutt for datateknologi, elektroteknologi og realfag ved Martin Fodstad Stølen, Institutt for miljø- og naturvitenskap ved Stein Joar Hegland og Truls E. Jørgensen ved HVL som skal bruke resultatet av prosjektet videre i sin masteroppgave.

Dette prosjektet er et delprosjekt av et større system hvor det skal utvikles en pakke for varsling og deteksjon av hjortevilt. Det andre delprosjektet, "*Autonomt kamerasystem for detektering og varsling av vilt ved vei*" utvikler sensorer og maskinvare for systemet. Det andre delprosjektet blir referert til som *maskinvare prosjektet* og deres produkt som *sensor system* i denne rapporten.

Delprosjektet som *denne* rapporten dekker omhandler utvikling og trening av en dyplæringsmodell og et programvaresystem som benytter modellen til å detektere og varsle ved detektering.

1.4 Problembeskrivelse og mål

Statens Vegvesen har i dag flere typer varsling for hjortevilt. Samtlige varslinger har problemer med at trafikanter ikke respekterer de og senker ikke farten ved varsel. Varslingssystemene som brukes i dag er for unøyaktige og har et høyt antall av falske-positive og negative varslinger, dette påvirker trafikantenes tiltro til systemet.

Målsettingen med prosjektet er å utvikle en modell for nøyaktig deteksjon og varsling av hjortevilt, ved bruk av dyplæring. Prosjektet skal også utforme en prototype for programsystemet som benytter modellen til å detektere hjortevilt og sende ut varsling ved deteksjon.

Modellen og programsystemet skal anvendes i et produkt sammen med sensorsystem bestående av kamera, maskinvare og varslingssystem. Produktet blir plassert på veistrekninger og/eller viltsluser med hjortevilt.

1.5 Oppbygging av rapporten

Denne rapporten inneholder 10 kapitler:

Kapittel 1 *En introduksjon og gjennomgang av problembeskrivelse og mål til prosjektet.*

Kapittel 2 *En detaljert bakgrunn av prosjektet hvor krav og avgrensninger for systemet vises.*

Kapittel 3 *En gjennomgang av prosjektets design, initiell løsning, prosjektets planlegging, verktøy og valgt løsning.*

Kapittel 4 *Gjennomgang av arkitektur til modell og programsystem, samt hvordan det blir utviklet.*

Kapittel 5 *Gjennomgang av evalueringsmetoder og evalueringsresultat for modell og system.*

Kapittel 6 *Diskusjon av fremgangsmåte, modell, programsystem, avgrensninger, og resultat.*

Kapittel 7 *Konklusjon og videre arbeid, her diskuteres det hvilke muligheter for videreutvikling prosjektet har.*

Kapittel 8 *En liste av prosjektets referanser.*

Kapittel 9 *Vedlegg.*

Kapittel 10 *Ordliste for fremmedord og deres betydning.*

2 PROSJEKTBESKRIVELSE

Kapittelet inneholder en forklaring av prosjektets bakgrunn og initielle krav. Deretter blir avgrensninger, tilgjengelige ressurser og eksisterende litteratur lagt fram.

2.1 Praktisk bakgrunn

2.1.1 Tidligere arbeid

Prosjektet tar utgangspunkt i funn og erfaringer fra studiet til Hegland og Frøyen (Hegland og Frøyen, 2021) hvor de undersøkte effektiviteten av dynamisk varslingssystem som varslet basert på tid, sted og sesong. Resultater fra fartsmålinger viser til liten eller ingen fartsreduksjon ved varsling av hjortevilt.

Et annet studie utført av Olsen og Holmberg i Sverige (Olsen og Holmberg, u.å.) viser likt at trafikanter har liten eller ingen fartsreduksjon ved varsling på en vei med fartsgrense på 80 km/t. Gjennomsnittsfarten er den samme selv ved aktiv varsling av hjortevilt.

2.1.2 Initielle krav

Det skal utvikles en modell og et prototype-programsystem for effektiv deteksjon av hjortevilt ved bruk av bilde eller video fra et sensorsystem plassert langs vei. I dette avsnittet presenteres krav til modell og programsystem.

Modellen skal:

- Kunne ta predikere deteksjon på data i form av bilder eller video.
- Modellen må kunne prosesseres i løpet av <2 sekunder.
- Modellen må detektere hjortevilt med høy presisjon.
- Modellen skal detektere på fargebilde
- Modellen skal kunne kjøre på svak maskinvare.
- Modell må kunne trenes opp til å gjenkjenne flere arter om nødvendig.

Programsystemet skal:

- Systemet må prosessere data i løpet av 2 sekunder.
- Systemet må varsle og detektere.
- Systemet må være lett å vedlikeholde.
- Systemet må kunne monitoreres for å vurdere ytelse.
- Systemet må kunne benytte modell på ekstern og lokal inndata.

2.1.3 Initiell løsnings-idé

Figur 2.1 viser en oversikt av systemet i brukermiljøet ved en viltovergang. Hjortevilt går innenfor kameraets synspunkt. Bildet blir deretter sendt til lokal maskinvare for analysering. Om modellen detekterer hjortevilt aktiveres et varsellys for trafikanter.

Systemet skal kunne kjøres på maskinvare med begrensede ressurser, eksempelvis Raspberry Pi som er en enkel og kostnadseffektiv modulær datamaskin (*Raspberry Pi Foundation, 2019)*. Systemet skal kunne utnytte prosessor (CPU) og/eller grafisk prosessor (GPU).

Det er forventet at systemet har tilgang på internett med varierende båndbredde og mobildekning, samt strømforsyning. Det er planlagt å benytte seg av en eksisterende algoritme for objekt detektering, utviklet i Python (*Python 3.10.4 documentation, u.å.)*. Etter å ha trent modellen skal den kunne eksporteres som en ferdig kompilert versjon som kan importeres og kjøres på forskjellig maskinvare.

Programsystemet vil kontinuerlig lytte på og prosessere inndata fra et kamera på den trente modellen i sanntid. Om programsystemet detekterer hjortevilt genereres og sendes et varsel til varslingssystem.

Figur 2.1: Systemet visualisert i brukermiljøet

2.2 Avgrensninger

Prosjektet tar utgangspunkt i at problemstillinger knyttet til valg av type kamera, kvalitet på bilder og valg av maskinvare ikke inngår i dette prosjektet og blir utformet av maskinvareprosjektet.

For å få dannet et godt nok datasett til å trene dyplæringsmodellen på har det blitt satt noen avgrensninger:

- Det kun blir fokusert på *hjort* under kategorien hjortevilt grunnet det blir lettere å bygge et bedre treningssett med et avgrenset fokus.
- Modellen skal trenes på fargebilder fra et normalt kamera da det er relativt få bilder av hjort fotografert med termisk eller infrarødt tilgjengelig i det offentlige domenet.

Dette og maskinvare prosjektet opererer med ulike fremdriftsplaner og frister. Dette gjør det utfordrende å koordinere arbeidet mellom prosjektene. Derfor gjort en avgrensning i samråd med oppdragsgiver om at prosjektene skal operere selvstendig med enkelte møter i løpet av prosjektenes forløp.

Prosjektene samarbeider under prosjektenes gjennomføring og deler informasjon og diskuterer valg av teknologier. Det lar seg ikke gjøre med reell testing i felt som del av dette prosjektet, ettersom hjortevandringen er over før prototypen ferdigstilles. Testing og evaluering gjennomføres derfor ved bruk av simulerte tilfeller bestående av bilder eller videoopptak tilnærmet sannsynlige og reelle situasjoner.

2.3 Ressurser

Effektiv utvikling og testing av modeller krever maskinvare i form av CPU eller dedikert GPU. Prosjektets forfattere har privat maskinvare med CPU- og GPU kraft nok til å trene modellene effektivt. Det er derfor ikke nødvendig å søke om midler til å anskaffelse av maskinvare.

Trening av modellen krever store mengder bilder av hjort. Bilder levert av oppdragsgiver Stein Joar Hegland, Alf Erik Røyvik og Krister Seljeset blir brukt til å trene modellen. Bildene er hentet fra privat samling.

For at bildene skal kunne brukes i trening av modell må bildene markeres på en måte slik at algoritmen vet hva den ser etter. Her blir verktøyet *makesense.ai (Make Sense, u.å.)* brukt. Nettsiden lar bruker laste opp bilder og markere objekter i bildet med koordinater som lar algoritmen *se* objektet.

Når bilder har blitt markert kan modellen trenes opp og er klar til å bli kjørt. Python 3.10 (*Python 3.10.4 documentation, u.å.*), PyTorch (PyTorch, u.å.) og CUDA toolkit (CUDA Toolkit, 2013) er nødvendig for å kjøre en modellen lokalt på Windows 10.

For å øke produktivitet blir treningsdata lastet opp og algoritmen kjørt i Google Colab (Google Colaboratory, 2019), et verktøy som lar hvem som helst skrive og eksekvere Python kode i nettleser. Google Colab gir brukere tilgang til skytjenester hvor grafikkprosessor kraft kan lånes.

Andre ressurser inkluderer oppdragsgivers kontaktperson Martin Fodstad Stølen og masterstudent Truls Erlend Jørgensen som bidrar med ekspertise til å forme deler av prosjektet.

Professor i økologi Stein Joar Hegland med ekspertise om hjortevilt som også har bidratt med bilder til datasett og tidligere forskning om problemstillingen.

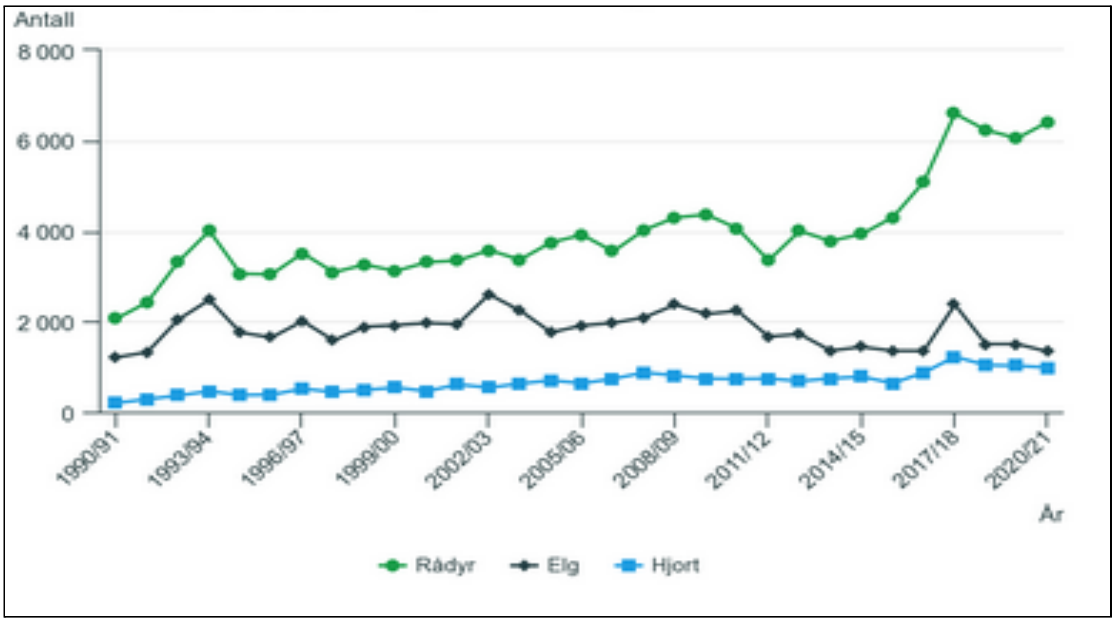
Reza Arghandeh med ekspertise på maskinlæring og algoritmer.

Per Christian Engdal, intern veileder ved HVL med veiledning for prosjekt.

2.4 Litteratur om problemstillingen

2.4.1 Påkjørsel av hjortevilt

Påkjørsel av hjortevilt er et stort trafikksikkerhetsmessig problem og forårsaker store skader og lidelser på dyr og mennesker. Dette er en stor samfunnsutfordring som blir større i takt med økende trafikkvolum og videre utbygging av infrastruktur som vist i figur 2.2.



Figur 2.2: Statistikk over påkjørsler av hjortevilt fra 1990-2021 (SSB., u.å.)

Ved testing er det avdekket at lengre, åpne strekker kan resultere i færre påkjørsler da vilt er lettere å se (Hegland og Hamre, 2018). Hegland og Hamre skriver videre at dersom dette ble tatt hensyn til under utbygging av infrastruktur kan antallet påkjørsler reduseres. Det ble og påvist større sannsynlighet for at hjortevilt krysser veien under mørke tilstander da de hviler på ett sted og forflytter seg om natten for å spise.

De eksisterende løsningene på problemet er ikke så effektive som de ideelt skulle ha vært. Det nyeste forsøket på en løsning var da trafikanter ble varslet via lys når det beregnet sannsynlig at hjortevilt var til stede ved veibanen *(Dynamiske hjorteskilt i Vestland fylke, 2017-2021)*. Det ble ikke registrert faktisk hjortevilt ved veibanen når varselet ble sendt ut, noe som førte til høyt antall falske- positive og negative varslinger. Over tid førte dette til at trafikantene tolket varslingen som falsk alarm og unnlot å redusere farten (Hegland og Frøyen, 2021). Basert på tidligere undersøkelser er det derfor tydelig at det kreves en bedre og mer nøyaktig løsning som trafikanter kan stole på.

2.4.2 Objekt deteksjon og dyplæring

I en tidligere masteroppgave ved NTNU av Kaarud, Nordvik og Paulsen, hvor de prøver å detektere sau ved bruk av algoritmer (Kaarud, Nordvik og Paulsen, 2020) var en lignende problemstilling løst ved hjelp av *You Only Look Once* (*YOLO)v3-tiny* algoritmen, i kombinasjon med termisk- og fargekamera.

YOLOv3-tiny er en objekt-detekterings algoritme som kan detektere et gitt objekt den er trent opp til med høy treffsikkerhet. Modellen er lett å trene opp og stiller lave krav prosesseringskraft under bruk.

Tabell 1 viser tall hentet fra Kaarud, Nordvik og Paulsen som viser resultat fra testing av algoritmene. *Mean average precision* (mAP) viser til gjennomsnittlig treffsikkerhet. Verdier fra 80% og opp er gode verdier. Merk at grenseverdi (threshold), altså hvor sikker modellen må være for å gjenkjenne objektet er lavt da det ikke er kritisk i deres tilfelle. Dette forårsaker et høyere antall falske positive deteksjoner som en ser helt til høyre i tabellen. Likevel er dette gode resultat.

Basert på Kaarud, og Paulsen sine gode resultater vil prosjektet vurdere å benytte en nyere variant av YOLO-algoritmen.

Version:	mAP:	Threshold	White:	Gray/brown:	Black:	Avg. false positives:
YOLOv3	98.37%	20%	95.5%	89.6%	92.1%	0.41
YOLOv3-tiny	96.60%	20%	96.5%	88.9%	95.2%	0.64

Tabell 1: Resultat fra masteroppgave hvor eldre versjon av YOLO algoritmen ble brukt. (Kaarud, Nordvik og Paulsen, 2020)

3 DESIGN AV PROSJEKTET

Dette kapittelet vil gå over mulige løsninger, verktøy, vurdering av risiko og plan for framdrift og evaluering.

3.1 Forslag til løsning

Løsningen for deteksjon av hjort krever høy presisjon og mulighet for å prosessere data på maskinvare med begrenset prosesseringskraft. Valg av egnet algoritme for dyplæringsmodellen vil være viktig for å oppnå ønskede resultat. I dette kapittelet presenteres, vurderes og diskuteres ulike dyplæring algoritmer for bruk i dette prosjektet.

3.1.1 You Only Look Once (YOLO)

You Only Look Once (YOLO) algoritmene er best kjent for hvor effektive de er i kombinasjon med prosesseringstid. Grunnen til at algoritmen er effektiv er delvis fordi den *ser* på bildet én gang for å prosessere det.

3.1.2 Single Shot Detection (SSD)

SSD (Single Shot Detector) fungerer likt som YOLO, hvor algoritmen *ser* bildet kun én gang før prosessering. Dette gjør at både SSD og YOLO er veldig effektive.

3.1.3 Faster Region based Convolutional Neural Network (Faster R-CNN)

Faster R-CNN (Faster Region based Convolutional Neural Network) er en algoritme som ser over et bilde flere ganger. Som resultat av dette stiller den betydelig høyere krav til maskinvare men har veldig god nøyaktighet. Dette gjør den godt egnet til applikasjoner hvor et raskt svar ikke er kritisk.

3.1.4 Egenutviklet algoritme

En annen alternativ løsning er at prosjektet utvikler en algoritme selv. Dette vil gjøre at prosjektet potensielt kan ende opp med en svært godt egnet algoritme, men vil være svært komplisert og tidkrevende.

3.1.5 Diskusjon av alternativene

Å bruke en eksisterende algoritme virker som den beste løsningen med utgangspunkt i prosjekt forfatternes kompetansenivå og prosjektets tidsramme. Da vil prosjektet kunne bruke lengre tid på å optimalisere en eksisterende algoritme mot målet ved valg og prosessering av bilder samt justering av parametere.

Mens det er muligheter til å lage en egenutviklet algoritme er det vanskelig å vite om sluttresultatet faktisk vil være bedre enn de allerede eksisterende algoritmene. Av de eksisterende algoritmene er de alle gode, men ikke nødvendigvis til dette brukstilfellet.

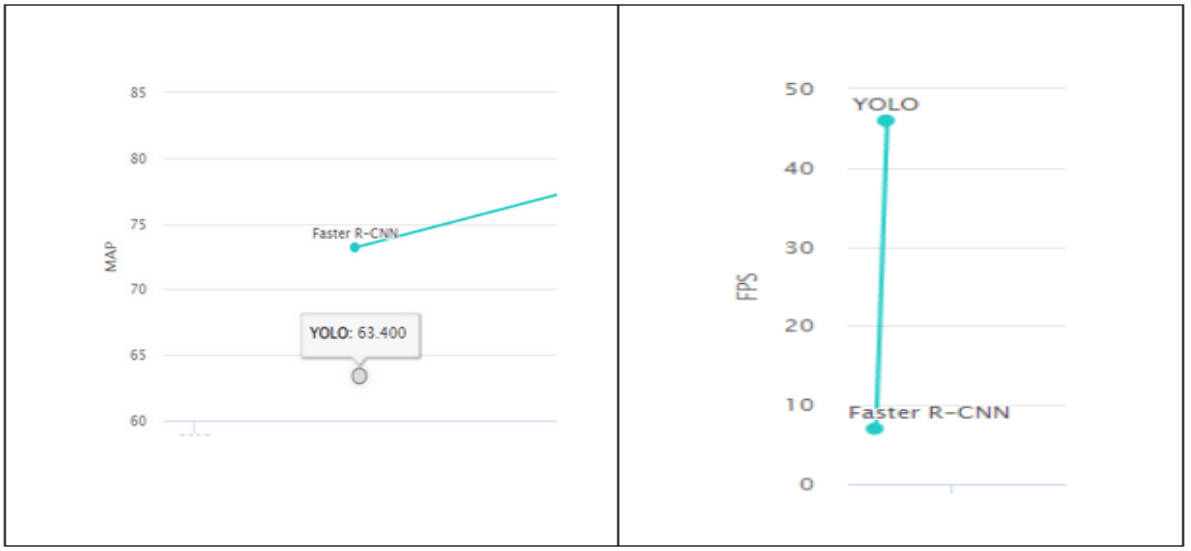
Faster R-CNN, heretter kalt *FRCNN* er mer nøyaktig, men tregere algoritme. Det vil være kritisk at algoritmen detekterer hjorte raskt nok da sen varsling kan føre til kollisjon.

Grunnen til at den er mer nøyaktig er at den ser over bildet flere ganger, men dette forårsaker større ventetid for analysering for hvert bilde *(Nguyen et al., 2020; Srivastava et al., 2021).*

Figur 3.1 viser sammenligning av YOLO og FRCNN modellene hvor begge modellene har blitt trent på PASCAL VOC 2007, et treningssett med ca. 9900 bilder med 20 objekt klasser som personer, fugler, katt, hunder og hest.

På venstre side av figur 3.1 sees mAP altså gjennomsnittlig treffsikkerhet. FRCNN får en treffsikkerhet på 73% og YOLO 63%. På høyre side vises ytelse målt i bilder per sekund(FPS), FRCNN med 7 FPS er betydelig tregere enn YOLO med 46 FPS. Det er ukjent hvilken maskinvare testene ble utført på.

På begrenset maskinvare kan bilder per sekund bli enda lavere, hvor modellen bruker flere sekunder på et eneste bilde.



Figur 3.1: Viser ytelse på hastighet og nøyaktighet på YOLO og FRCNN (paperswithcode.com. u.å.)

Single Shot Detection heretter kalt *SSD* er en aktuell algoritme å ta i bruk da den er relativt rask. Det er uvisst hvilke oppløsning inndata vil være på og det kan variere på avstand mellom sensor og hjortevilt. Derfor kan det være en risiko å bruke SSD da den er avhengig av høy oppløsning og gode bilder for å gi gode resultater *(Nguyen et al., 2020; Srivastava et al., 2021).*

Kvalitativ analyse på forskjellige algoritmer, hvor hastighet og nøyaktighet ble testet *(Srivastava et al., 2021)* viser at SSD et godt alternativ men viser svakhet på nøyaktighet av mindre objekter, relativt til oppløsningen av bildet. SSD trenger og en del mer treningsdata enn de andre algoritmene, muligens mer enn det som er tilgjengelig på dette tidspunktet.

Figur 3.2 viser resultat fra varianter av SSD som er trent på *Common objects in context* (COCO) datasettet. COCO består av ca 330.000 bilder med vanlige gjenstander som biler, buss, hund, epler og liknende i normal kontekst. SSD300 og SSD512 viser til oppløsningen på bildene, altså 300x300 og 512x512. mAP ligger på mellom 41.2% og 46.5 mellom de to variantene hvor modellen som benytter seg av lavest oppløsning på bilder presterer dårligst.

Method	data	Avg. Precision, IoU: 0.5:0.95	0.5	0.75
SSD300	trainval35k	23.2	41.2	23.4
SSD512	trainval35k	**26.8**	**46.5**	**27.8**

Figur 3.2 (Hui, 2019). Utsnitt av resultat fra trening på COCO datasettet.

YOLOv5 har en god balanse av nøyaktighet og hastighet. Basert på utforskende testing utført i tidlig i prosjektet har *YOLOv5* algoritmen (*Ultralytics 2020)* vist seg til å være god på deteksjon til tross for at den manglet tilstrekkelig treningsdata. Kvaliteten på resultatet fra algoritmen er mest avhengig av mengde og kvalitet på data.

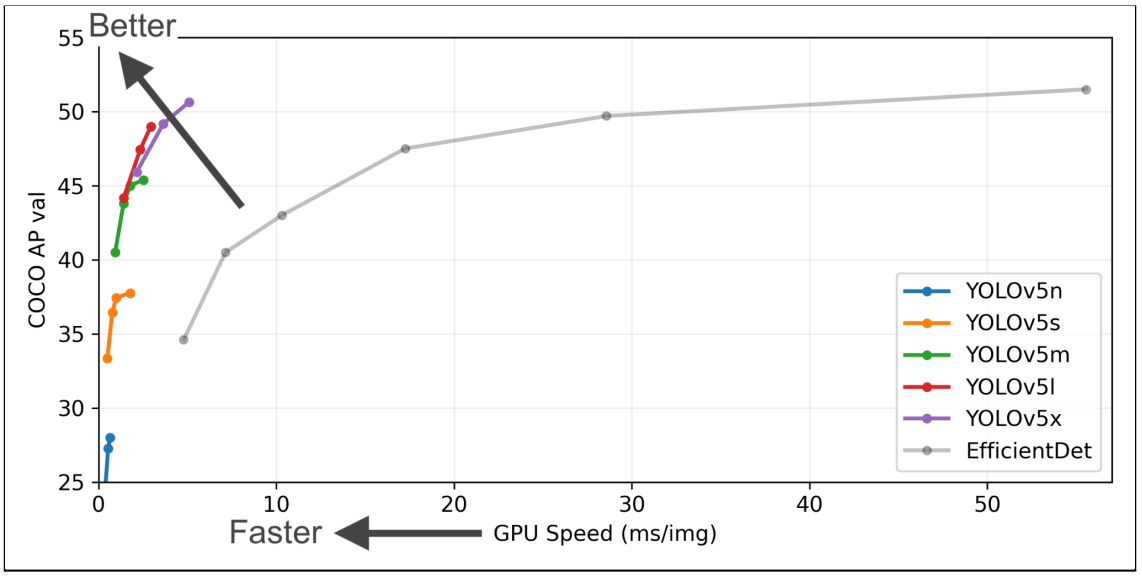
Det finnes flere varianter av YOLO algoritmen som vist i tabell 2. Nano, small og til en viss grad medium er optimalisert for maskinvare med begrensede ressurser. Large og Extra Large egner seg best til skybaserte løsninger hvor det benyttes kraftig maskinvare.

Modellene optimalisert for svak maskinvare ofrer nøyaktighet i bytte mot hastighet.

Modell	Beskrivelse
YOLOv5n - Nano	En veldig rask versjon med lavest forventet nøyaktighet. Denne passer godt til lokal kjøring på svakere maskinvare.
YOLOv5s - Small	Denne versjonen er fortsatt veldig rask, men er en del mer nøyaktig enn Nano varianten. Passer godt til lokal kjøring på svakere maskinvare.
YOLOv5m - Medium	Varianten er nå begynt å bli noe mer krevende med tanke på maskinvare. Høyere forventet nøyaktighet. Passer godt til lokal kjøring på svakere maskinvare som kan benytte kraft fra grafikkprosessor.
YOLOv5l - Large	Dette er en av de større variantene som er mer nøyaktig men stiller høyere krav til maskinvare. Anbefalt bruk i skybasert løsning.
YOLOv5x - Extra Large	Den tyngste varianten. Betydelig mer nøyaktig og krevende. Passer best til sterk maskinvare og skybasert løsning.

Tabell 2: Forskjellige versjoner av YOLOv5

Figur 3.3 viser forskjellene mellom de tidligere nevnte variantene. Modellene blir testet på COCO-treningssettet fra tidligere. Y-aksen viser til den gjennomsnittlige treffsikkerheten på de individuelle modellene i prosent, og X-aksen viser til hastighet per bilde som blir sett over. Høyere treffsikkerhet og lavere prosesseringstid er bedre.



Figur 3.3: En figur som viser forskjellen mellom de forskjellige versjonene. Modellene blir trent og testet på COCO datasettet (Ultralytics, 2020).

3.2 Valgt løsning

Basert på egen testing, evaluering av relevante algoritmer *(Nguyen, 2020; Srivastava, 2021)* og resultat fra tidligere arbeid (Kaarud, Nordvik, og Paulsen, 2020) tilsier at YOLO er den best egnede algoritmen for dette prosjektet. Prosjektet velger derfor å benytte YOLO-algoritmen.

Som tidligere nevnt under *kapittel 3.1.5* finnes det flere varianter av YOLO algoritmen. Dette lar prosjektet benytte en algoritme som kan tilpasses etter krav til treffsikkerhet og hastighet.

Prosesseringstid og dermed nøyaktighet kan videre forbedres med eksterne enheter som Coral. Coral er en USB-akselerator egnet til å forbedre prosesseringstid i maskinlæring- og dyplæringsapplikasjoner for enheter med svakere maskinvare (Google Coral, 2022).

3.3 Valg av verktøy

Det er behov for å ta i bruk en rekke verktøy for å gjennomføre prosjektet.

Prosjektet benytter:

- *Python* blir brukt som programmeringsspråk da Python har mange gode rammeverk for bruk i maskinlæring.
- *Visual Studio* og *IntelliJ* er oversiktlige utviklingsmiljø med støtte for Python som passer godt til dette prosjektet.
- *Github* lar gruppen dele kode og legger opp til samarbeid på forskjellige datamaskiner.
- *Google Colab* lar brukerne eksekvere krevende Python kode på et nettbasert utviklingsmiljø. Nettsiden lar brukerne låne prosessorkraft fra skytjenesten. Dette gjør at en kan teste algoritmer på hvilket som helst system uavhengig av maskinvare.
- *Makesense.ai* er valgt for å laste opp bildesett og sette koordinater på bildene. Koordinatene sier hva algoritmen skal *se* etter under trening. Etter markering kan tekstfiler som inneholder koordinater knyttet til et gitt bilde lastes ned. Algoritmen ser på objektet innenfor området og lærer basert på dette.
- *CUDA toolkit*, en programvare som lar grafikkprosessor ta over store deler av prosesseringen under trening, noe som vil akselerere prosessen da grafikkprosessorer er spesialisert og veldig effektive til å utføre beregninger av tall.
- *PyTorch* et åpen-kilde rammeverk egnet for prosessering ved dyp læring algoritmer. PyTorch blir brukt for å implementere objekt detektering basert på YOLO- algoritmen.
- *Trello* er en nettløsning som lar brukere lage Kanban brett som kan deles med andre.

3.4 Prosjektmetodikk

3.4.1 Utviklingsmetodikk

Prosjektet er ikke fullt ut et utviklingsprosjekt. Ettersom prosjektet har god tilgang til brukerrepresentanter betyr det at prosjektet ikke passer en agil utviklingsmetode som Scrum (*Schwaber, 2002*) i prosjektets faser.

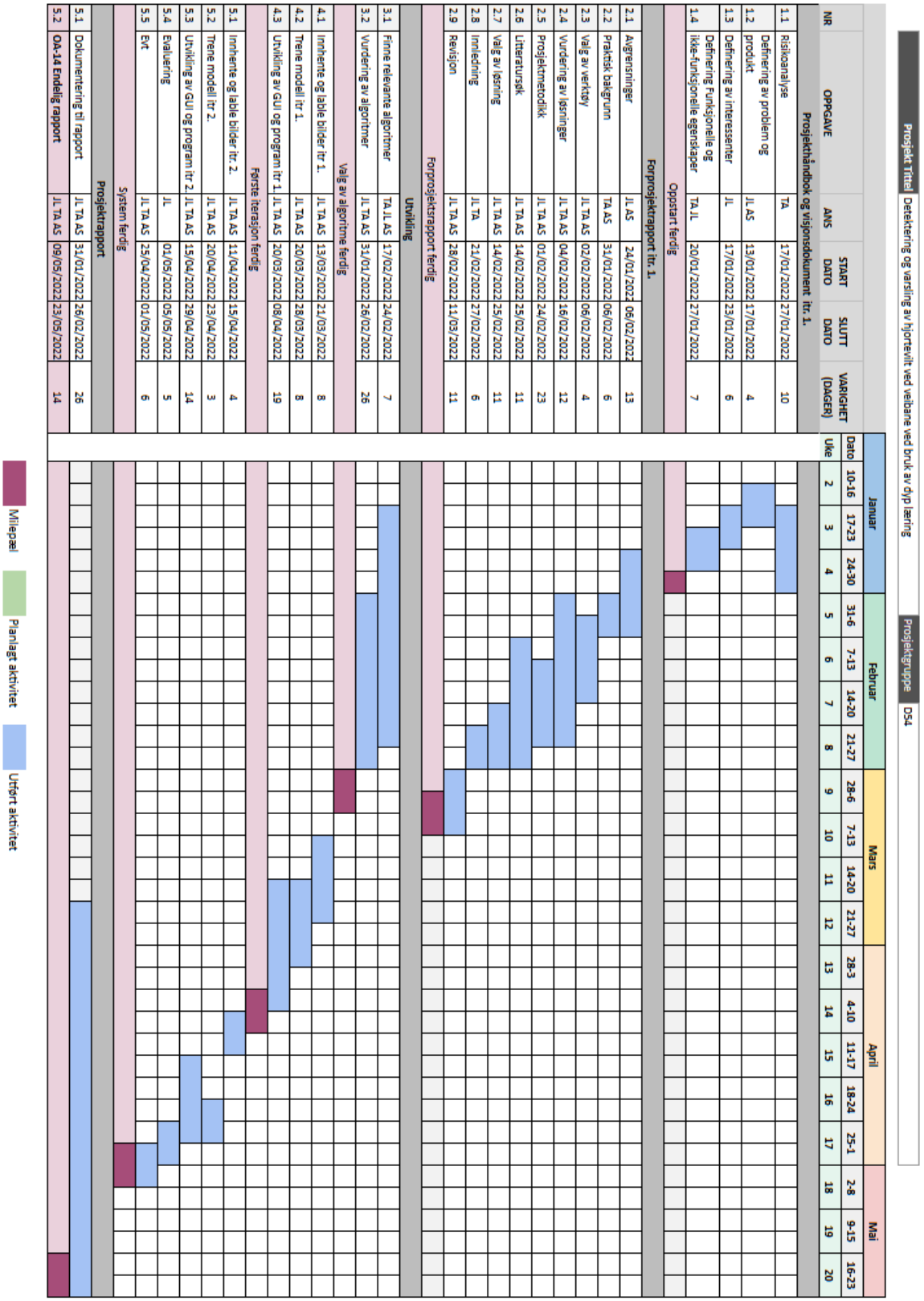
Da prosjektet har jobbet i varierende iterasjoner passer det mer naturlig å basere prosjektet på en iterativ utviklingsmodell. Prosjektet har varierende lengder på iterasjoner som illustrert i framdriftsplan i figur 3.4. Prosjektet tar i bruk enkelte framgangsmåter som *Kanban board* fra Kanban (*Hopp og Spearman, 2003*) og iterasjoner og møter fra Scrum (*Schwaber, 2002*).

Den første iterasjonen omfavner leveranse av forprosjekt og definerer problemområdet. Det settes og krav til løsning og løsningsdesign. Iterasjonen inneholder og trening og validering av den første versjonen av modell ved testing og valg av algoritme.

Deretter går prosjektet inn i en utviklingsfase bestående av flere korte iterasjonen hvor modellen trenes og evalueres. Samtidig blir programvaren utformet og testet. Prosjektet har tilnærmet daglige møter som hvor koordinering av aktiviteter og fremdrift blir diskutert. Kanban har blitt brukt gjennom Trello (Trello, u.å.) til å fordele oppgaver og aktiviteter.

Brukerinvolvering og testing begynner i utviklingsfasen, siden delprosjektene ikke har en felles prototype klar for testing blir dette simulert.

3.4.2 Prosjektplan



Figur 3.4: Framdriftsplan illustrert via et GANTT-diagram

Prosjektets fremdriftsplan er representert som et GANTT-diagram i figur 3.4 da dette er et kjent verktøy for å illustrere plan for prosjekt. Dette prosjektet har fire hovedmilepæler:

Oppstart ferdig	*Inkluderer første iterasjon av prosjekthåndbok og visjonsdokument.*
Forprosjektrapport levert	*Inkluderer avgrensninger, valg av løsning, litteratursøk.*
Modell og programsystem ferdig	*Inkluderer valg av algoritme, samt iterasjoner for trening av modell og utvikling av programsystem.*
Endelig prosjektrapport levert	*Inkluderer ferdigstilling av prosjektrapport inklusive støttedokumenter.*

3.4.3 Risikovurdering

I tabell 3 under blir forskjellige risikoer som kan resultere i negative konsekvenser identifisert i prosjektet analysert Alle potensielle problem evalueres og forsøkes håndteres der det er mulig (Se Vedlegg 2: Prosjekthåndbok).

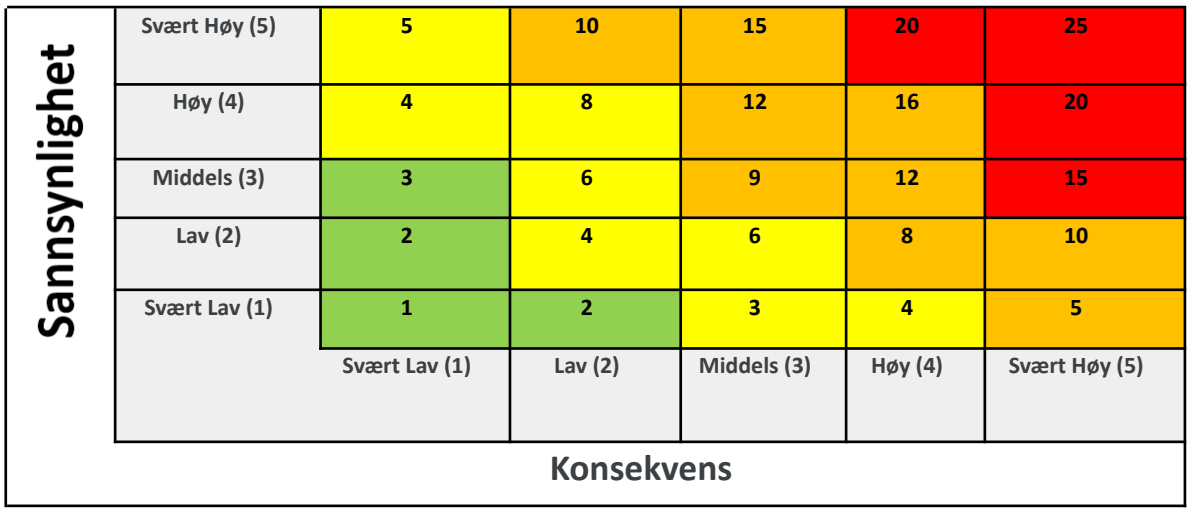
Lavere risiko produkt er bedre.

Nr	Hendelse	Årsak	Sann-synlighet	Konse-kvens	Risiko-produkt	Tiltak
1	Modell gir falsk positiv eller falsk negativ.	Treningsdata er ikke god nok, og modellen er overtilpasset eller ikke generalisert nok.	Lav(1)	Høy (4)	4	Se på hvordan modellen presterer, oppdatere/forbedre treningsdata slik at resultat blir mer nøyaktig. Sette krav på høyere estimering; f.eks alt under 60% treffsikkerhet blir sett på som ikke godt nok, for å forhindre falsk positiv.

Nr	Hendelse	Årsak	Sann-synlighet	Konse-kvens	Risiko-produkt	Tiltak
2	Modell får dårlig nøyaktighet etter ytterligere trening av modell.	Dersom modellen skal utvides og må kunne detektere mer, trenger modellen mer inndata. Under utvidelse kan modellen bli trent opp dårlig og dermed få redusert nøyaktighet.	Lav (2)	Høy (4)	8	I forkant kan gruppemedlem passe på at all inndata er god. Skulle det hende at modellen fikk dårlig trening kan modellen bli trent opp på ny.
3	Modell mangler stor nok mengde treningsdata.	Det er for få kjente ressurser til å få god trening, og gruppen finner ikke eksisterende datasett.	Høy (4)	Høy (4)	16	Bruke mer tid på å finne relevante ressurser på nett, og kontakte personer som kan ha relevante ressurser tilgjengelig.
4	Gruppen rekker ikke å levere et resultat.	Dårlig tidsforbruk eller uforutsette utfordringer kan føre ta opp mye tid, og forsinke prosessen betydelig.	Lav (2)	Svært høy (5)	10	Planlegge grundig hvordan arbeidet skal skje fremover, og eventuelt ta kontakt med veiledere angående eventuell utfordring.
5	Gruppen klarer ikke å implementere kritiske funksjoner eller nå initielle krav.	Manglende forkunnskaper til å kunne få et fullstendig resultat.	Lav (2)	Svært høy (5)	10	Ha god dialog med oppdragsgiver og setter begrensinger for prosjektet hvor det er nødvendig, for at et resultat kan bli levert.

Nr	Hendelse	Årsak	Sann-synlighet	Konse-kvens	Risiko-produkt	Tiltak
6	Gruppen mangler tilgang til riktig verktøy, maskinvare/programvare	Enkelte programvarer og maskinvarer koster ofte penger, og kan være kritisk til å kunne produsere et godt resultat.	Middels (3)	Svært høy (5)	15	Diskutere med oppdragsgiver om hvilke verktøy som er nødvendige, og søke om midler til å kunne skaffe verktøyet. Alternativt kan gruppen se etter andre gratis verktøy som kan oppnå tilnærmet ønskede resultat.
7	Manglende evaluering av prosess og resultat, som kan føre til feil resultat.	Gruppen har ikke god nok kommunikasjon med oppdragsgiver og veileder og kan ende opp med å levere feil resultat.	Lav(2)	Høy(4)	8	Være mer oppmerksom på å holde jevnlige møter med oppdragsgiver og veileder for å oppnå forventet resultat.
8	Programvare kan stoppe på grunn av en feil.	Programvare blir lansert med en ukjent feil.	Middels (3)	Lav (2)	6	Teste programvare grundig før lansering og se til at alt fungerer som det skal.
9	Programvare kan slutte å fungere på grunn av oppdatering etter installering.	Ved oppdatering av programvare eller operativsystem kan algoritmen slutte å fungere.	Middels (3)	Lav (2)	6	Koordinere eventuelle oppdateringer slik at det ikke blir stor nedetid på programvare. Det tar ikke lang tid å starte opp programvaren igjen.

Tabell 3: Risikovurdering for prosjektet



Figur 3.5: Risikomatrise med vekter som analysen er basert på.

Risikoer har blitt vurdert på et risikoprodukt som er beregnet på grunnlag av:

*Sannsynlighet * Konsekvens = Risikoprodukt*

I figur 3.5 sees skalaen for risikovurdering fra Skjølsvik og Voldsund som blir tatt i bruk. Sannsynlighet og konsekvens blir rangert fra 1-5 hvor 5 er høyest risiko (Skjølsvik og Voldsund, 2016, p.344).

Etter å ha identifisert hvilke risikoer som er ansett som mest relevante for prosjektet, blir prosessen planlagt med hensyn til risikoanalysen. Dette er for å minimere sannsynlighet for negativ innvirkning fra hendelsene.

3.5 Evalueringsplan

3.5.1 Plan for evaluering av modell

Det er forskjellige metoder å evaluere objekt detekterings modell. Vanligst er det å evaluere modellen på *presisjon* og *kompletthet*, *pixel accuracy* eller *Jaccard Index* og deretter beregne mAP (Padilla, Netto og da Silva, 2020).

I dette prosjektet blir modellen derfor evaluert på mAP under utvikling av modellen for å undersøke om det er forbedring eller regresjon av resultat. mAP blir evaluert ved å teste modellen på et valideringssett av bilder hvor innholdet er kjent, men modellen ikke har sett det før.

Som mål skal modellen ha en mAP på 80% eller høyere.

3.5.2 Plan for evaluering av programvare

Evaluering for programvaren blir gjennomført ved systemtest, brukertest og ytelsestest.

Systemtest innebærer testing av implementerte enheter/funksjoner etterhvert som de blir lagt til. Enheten testes for korrekt oppførsel under bruk og håndtering av eventuelle feil. Som mål skal systemtesten være feilfri.

Brukertest innebærer at oppdragsgiver bidrar med videomateriale til programsystemet og observerer funksjonalitet, ytelse og resultat. Oppdragsgiver gir tilbakemelding etter brukertest. Som mål skal oppdragsgiver være svært fornøyd.

Ytelsestest består av å måle gjennomsnittstid brukt på å prosessere et antall bilder. Som mål skal programsystemet prosessere bruke maksimum 2 sekunder per bilde, ideelt 1 sekund eller mindre.

4 DETALJERT LØSNING

I dette kapittelet dekkes metoder for å gjennomføre prosjektet. Det gis en oversikt over løsningen og prosessen samt en konseptuell oversikt over programsystem og arkitektur.

4.1 Modell

4.1.1 Modellutvikling

For at modellen skal bli pålitelig og nøyaktig er det viktig at modellen trenes riktig. Dette gjøres ved å bruke treningsdata hvor innholdet på bildene i størst mulig grad representerer brukstilfellet i form av lokasjon, vinkel og miljø.

Modellutviklingen tar utgangspunkt i en *pipeline* hvor en gjennomfører hvert steg i prosessen illustrert i figur 4.1.

Figur 4.1: Steg i prosessen for å trene YOLOv5 modellen.

4.1.2 Innsamlinger av bilder

I første iterasjon mottar prosjektet en rekke private bilder av hjort fra Stein Joar Hegland, Alf Erik Røyrvik og Krister Seljeset. Bildene inneholder hjortevilt i varierende situasjoner og vinkler. Selv om settet består av få bilder er det tilstrekkelig for å få en indikasjon på modellens ytelse og nøyaktighet.

I andre iterasjon øker mengden treningsdata etter ved bruk av programsystemets implementerte funksjon hvor det lagrer bilder av deteksjoner. Bilder hentes fra relevante videofiler eller strømmer med potensiell treningsdata ved å aktivere funksjonen og sette konfidensverdien lavt.

4.1.3 Forberedelse av data

Før data kan bli brukt til trening behandles bildene for å gi best mulig resultat. Dette skjer i form av speiling, kopiering og sanitering av bildene.

Alle svarthvitt bilder fjernes treningsdata for å forhindre at modellen trenes på bilder fra situasjoner der den ikke skal benyttes.

Bilder hvor hjort er svært nærme kamera fra uvanlige vinkler er ikke representativt for framtidig brukstilfelle. Derfor fjernes de. På grunn av saniteringen blir det initielle treningssettet i første iterasjon svært lite.

For å øke omfanget av tilgjengelig data blir bilder speilvendt og kopiert. Dette resulterer i at treningssettet blir tilnærmet doblet i omfang til å inneholde 185 bilder. Dette bidrar til å legge til rette for bedre trening av modellen.

Bilder hvor det ikke er hjort men består av bakgrunn med skog og natur blir tatt med for å trene modellen på hva den *ikke* skal se på. På denne måten bruker en de tilgjengelige data så mye som det lar seg gjøre.

Før bildene er klar til trening markeres alle bilder. Bildene blir lagt inn i *makesense.ai (makesense.ai .u.å.)* hvor hjort på bilder markeres manuelt. For hvert bilde med markeringer blir en tekstfil med koordinater for plassering på hjort(ene) generert. Bilder uten hjort, får en tom tekstfil uten koordinater.

4.1.4 Trening av modell

Når alle bildene og tekstfilene er klare deles filene i et treningssett som består av 80% av datasettet og et valideringssett som består av resterende 20%. Valideringssettet er brukes til å teste modellen på ny og ukjent data.

Før trening legger en inn verdier for parametere som bildestørrelse i form av oppløsning, hvilken variant av modell som skal trenes og hvilke bilder som modellen skal trenes på. De forskjellige modellene trener på bildene i opptil iterasjoner 3000 ganger med tidlig stopp, da stopper trening av modell automatisk dersom det ikke er forbedring i løpet av siste 100 iterasjoner.

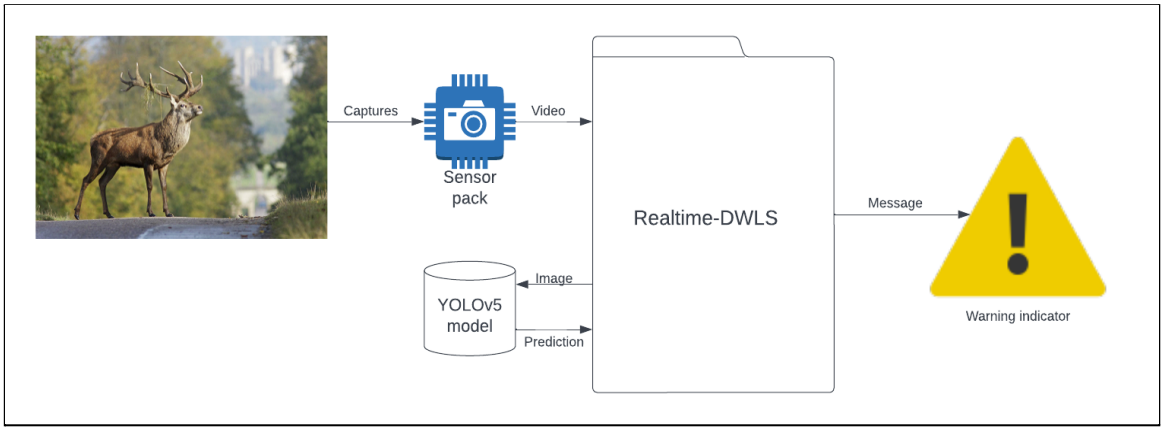
Eksempelvis ble størrelse på bilder endret for å finne best egnet bildestørrelse for modellen ut fra datasettet. Tabell 4 viser resultatene fra forskjellige treninger med forskjellig størrelse på bilder. Med datasettet i andre iterasjon gir størrelsen 640x640 best resultat.

Modell	Bilde størrelse	Precision	Recall	mAP@0.5	mAP@0.5 :.95
YOLOv5 Small	240x240	0.900	0.774	0.824	0.499
	360x360	0.914	0.833	0.883	0.540
	480x480	0.915	0.841	0.886	0.563
	640x640	0.897	0.865	0.898	0.564

Tabell 4: Utvikling av resultater ved endring i bildestørrelse

4.2 Programsystem

4.2.1 Konseptuell oversikt



Figur 4.2 Viser en konseptuell oversikt over programsystemet.

Figur 4.2 viser en konseptuell oversikt over programsystemet. Systemet får inn inndata fra sensorsystemet som ved bruk av kamera filmer omgivelsene. Programsystemet konverterer videostrømmen til bilder i sanntid og utfører en prediksjon for hvert bilde.

For hver prediksjon blir det sendt ut en melding med innhold om resultatet av predikering via en meldingstjeneste. Ved deteksjon tegner programsystemet avgrensningsbokser rundt hvert objekt før det blir sendt til det grafiske grensesnittet.

4.2.2 Funksjonalitet

For at programsystemet skulle kunne utføre oppgaven om å konvertere inndata i form av videostrøm i fargebilder til varsling i det aktuelle bruksområdet utvikles det til funksjonalitet basert på Use casene som finnes i *Vedlegg 3: Kravdokumentasjon*.

Use case: *Valg av kilde for sensor/kamera*

Bruker velger en kilde for inndata til programsystemet ved å trykke på en knapp i det grafiske grensesnittet.

Use case: *Valg av modell fil*

Bruker velger en kilde for hvilken modell som skal benyttes av programsystemet til å predikere på inndata. Modell fil velges ved å trykke på en knapp i det grafiske grensesnittet.

Use case: *Deteksjon*

Programsystem får inn inndata i form av video, utfører en prediksjon på bildet og gir tilbakemelding om det blir detektert hjort i bildet.

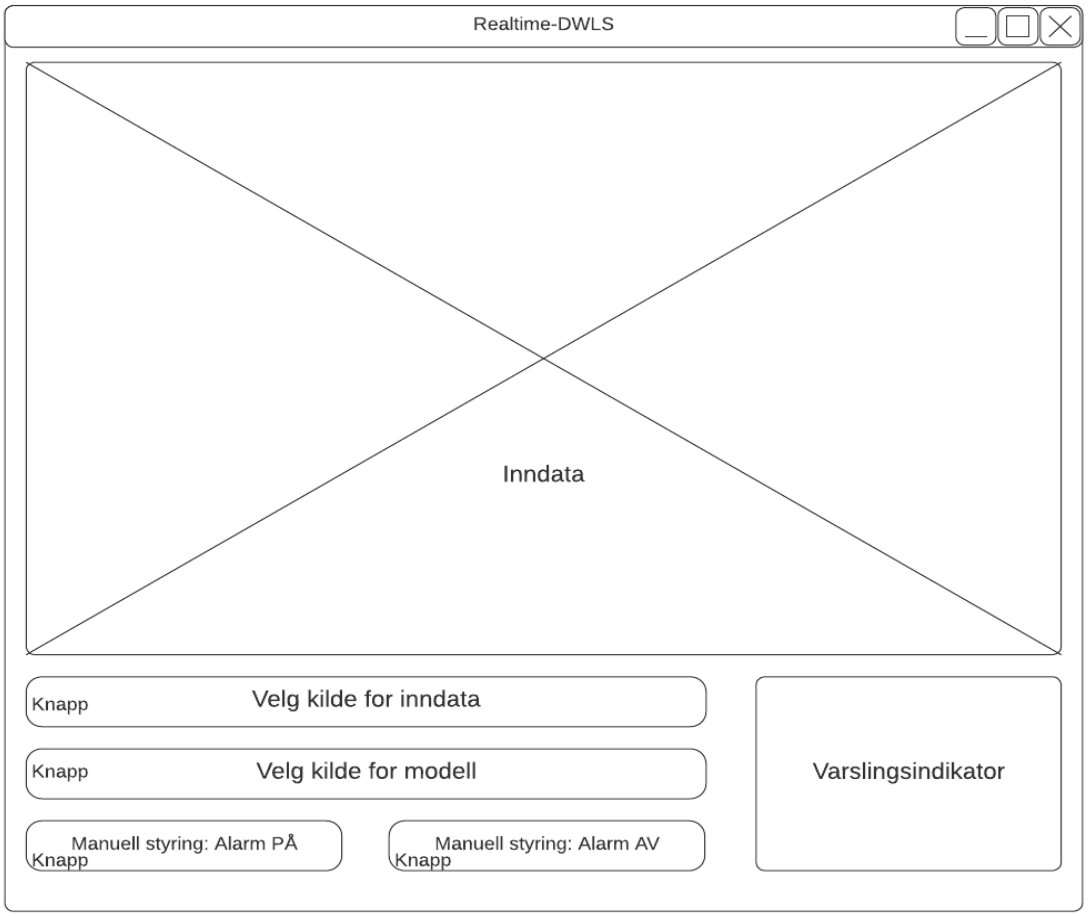
Use case: *Varsling*

Ved deteksjon blir det sendt ut varsling til gjennom varslingssystemet og bruker får beskjed om varsling visuelt.

Use case: *Valg av manuell varsling av eller på*

Ved drift av system kan varsling skrus av eller på manuelt ved behov, dette gjøres ved å trykke på knappen for Manuell varsling av eller på.

4.2.3 Brukergrensesnitt prototype

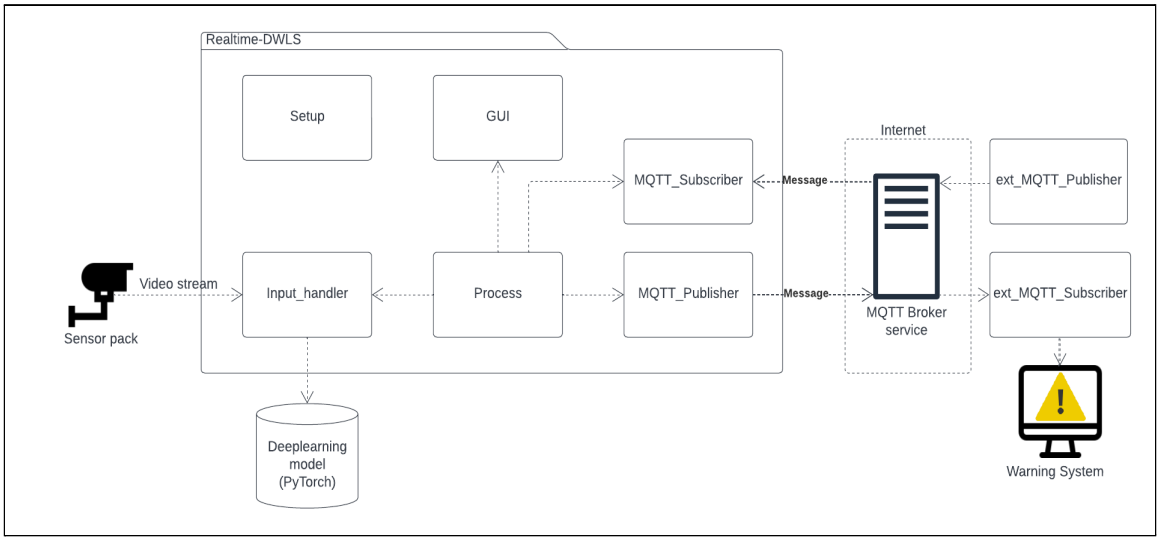


Figur 4.3: Wireframe for planlagt brukergrensesnitt

Figur 4.3 viser hvordan det grafiske grensesnittet er tiltenkt med knapper for valg av kildedata, manuell overstyring av alarm hvor varsling kan skrus av eller på etter behov.

Det er og tiltenkt en varslingsindikator som visuelt viser varsling ved deteksjon. I inndata feltet vises video etter at prediksjon har blitt utført og består av video med opptegnede avgrensningsbokser ved deteksjon.

4.2.4 Arkitektur



Figur 4.4: Oversikt over program systemets arkitektur (Se vedlegg 4, Systemdokumentasjon)

Programsystemet består av syv hovedkomponenter med hvert sitt ansvarsområde:

Input_handler mottar videostrøm fra sensor system og utfører prediksjon ved bruk av dyplæringsmodell gjennom rammeverket

Deeplearning model(PyTorch) er den opptrente YOLOv5 modellen som blir benyttet ved bruk av PyTorch rammeverket (PyTorch.org, u.å.).

Process håndterer operasjoner i programmet, henter bilde fra input_handler, sender det til prediksjon og sender resultat til *GUI* eller *MQTT_Publisher*

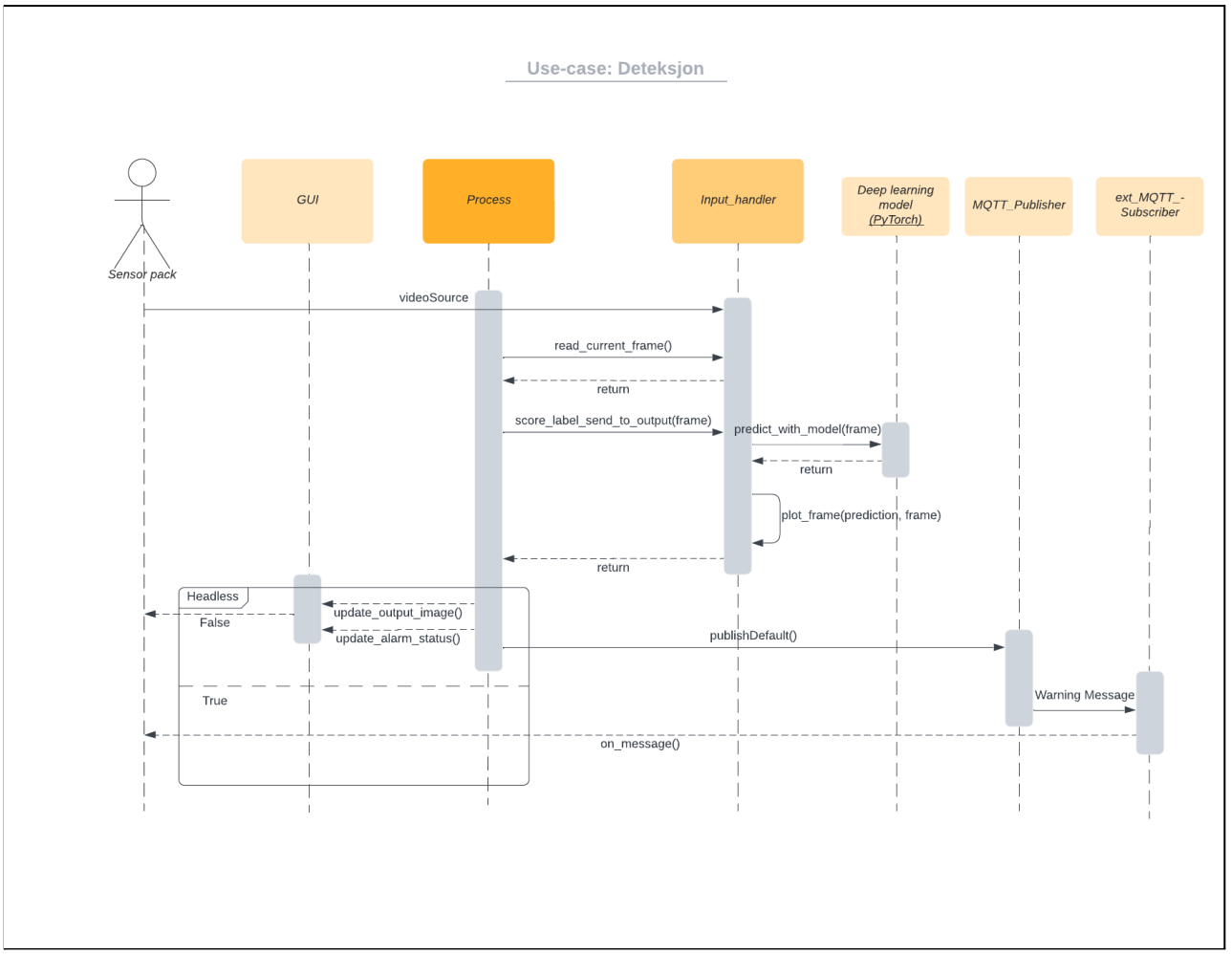
GUI representerer det grafiske grensesnittet for programsystemet og oppdaterer bilde og varslingsindikator basert på data fra *Process*.

Setup inneholder innstillinger for oppstart av program samt innstillinger for hvordan det grafiske grensesnittet skal vises.

MQTT_Publisher får data fra *Process* som settes sammen til en melding formatert som JavaScript Object Notation(JSON), JSON-meldingen blir sendt til abonnent.

MQTT_Subscriber tar inn melding fra ekstern en ekstern MQTT Publisher. I dette programsystemet venter den på en URL til en ny oppdatert modell som programsystemet laster ned ved mottak.

Når programsystemet starter initieres en instans av *GUI* og *Process* modulene. Process modulen instansierer en *Input_handler* som tar inn en videostrøm, samt *MQTT_Publisher* og *MQTT_Subscriber* for å håndtere meldinger inn og ut av programmet. Prosessen lytter og på innkommende meldinger via *MQTT_Subscriber* og kan få en ny URL til en oppdatert modell som den vil laster og setter som ny standard modell.



Figur 4.5: Sekvensdiagram (Se Vedlegg 4: Systemdokumentasjon) for brukstilfellet ***Deteksjon*** *(Se Vedlegg 1: Visjonsdokument)*

Figur 4.5 viser programflyt for brukstilfellet *Deteksjon*, Input_handler får inn inndata fra en video strøm. Process henter ett enkelt bilde fra videostrømmen og kjører bildet gjennom prediksjon funksjonen i Input_handler som videre benytter seg av PyTorch rammeverket til å utføre en prediksjon ved bruk av den opptrente YOLOv5 modellen.

Prosess får tilbake predikerte verdier og bilde fra Input_handler og sender så ut det nåværende bildet til GUI. Samtidig sender Process verdier som består av deteksjon, konfidensverdi, antall deteksjoner, lokasjon og klokkeslett fra prediksjon til MQTT_Publisher.

MQTT_Publisher sender ut en tekststreng formatert til JSON til en ekstern abonnent som initierer varsling på ekstern enhet. GUI initierer varsling på lokal enhet.

4.2.4.1 Komponent: Process

Ansvaret for komponenten består i å fungere som hovedleddet i programmet mellom inndata og utdata. Komponenten er en tråd som får inn data fra input_handler og legger det i en kø i påvente av at det blir sendt ut ut til bruker via GUI i form av visuell data og/eller MQTT_Publisher i tekst med JSON-format.

For at komponenten skal kunne kjøre samtidig som det grafiske grensesnittet har komponenten blitt lagt til som en egen tråd. Tråden kjører uavhengig av det grafiske grensesnittet for å unngå at trådene låses og blir stående å vente på hverandre.

Det isoleres ett enkelt bilde fra videostrøm om gangen som blir sendt til funksjonene som utfører prediksjon og plotting av avgrensningsbokser.

Under i figur 4.6 vises et forenklet eksempel av komponentens *run* metode, metoden som kjøres av tråden for hver oppdatering. Metoden sjekker om det er endring i inndata og utfører metodekall for å hente det nyeste bildet fra video strømmen. Bildet sendes så til funksjonen som utfører en prediksjon. Process får resultatet tilbake, som blir sendt det grafiske grensesnittet og/eller en MQTT melding som går ut av systemet.

```
def run(self):
    while (True):
        newModel_fromRemote = self.mqtt_subscriber.get_mqtt_source()

        if nyKilde is not None:
            setNyKilde()

        ret, current_frame = les_nyeste_bilde_fra_video()

        if(ret == False):
            feilmeldingIngenInndata

        if inndata and grafiskGrensesnitt:
            leggIkø.score_label_send_to_output(current_frame)
        if inndata and not grafiskGrensesnitt:
            score_label_send_to_output(current_frame)

        mqtt_publisher.publiserMeldingMedData()

        ventTilNesteBilde()
```

Figur 4.6: Process modulens run metode.

Videre viser koden under i figur 4.7 en forenklet versjon av hvordan prosessens metode benytter seg av funksjoner i Input_handler til å utføre en prediksjon og få tilbake data som skal sendes til det grafiske grensesnittet og/eller en MQTT melding:

```
def score_label_send_to_output(self, current_frame, rawFrame, gui):


    labels, cord = input_handler.prediker(current_frame)

    frame,
    detected_flag,
    detectedCount,
    lowestConfidence,
    highestConfidence=input_handler.plot_frame(prediction
current_frame)

    self.currentTime = tiden_nå()
    self.set_detected(detected_flag)
    self.set_detectedCount(detectedCount)
    self.set_confidenceValue(lowestConfidence, highestConfidence)


    if grafisk_grensesnitt:
         Oppdater_grafisk_grensesnitt()
```

Figur 4.7: Process modulens metode for å starte prediksjon.

4.2.4.2 Komponent: Setup

Ansvaret til Setup komponenten er å gjøre tilgjengelig et forhåndsdefinert oppsett for det grafiske grensesnittet under generell drift av programmet, samt manuelt oppsett ved oppstart.

Komponenten består derfor av to forskjellige deler hvor ansvarsområde er delt opp for grafisk brukergrensesnitt ved drift (*gui_setup*) og oppstart (*startup_setup)*. Setup komponenten har og ansvar for å supplere sti til inndata for modell og video strøm.

4.2.4.3 Komponent: Input_handler

Ansvaret for komponenten består i å håndtere inndata i rå form samt utføre operasjonene for prediksjon og tegning av avgrensningsbokser på bildet. Komponenten har og ansvar for lagring av bilder.

For å kunne utføre operasjoner på videostrømmen benyttes rammeverket OpenCV (OpenCV Introduction, u.å.). Videostrømmen blir lagt til som et OpenCV video objekt hvor ett og ett bilde hentes ut fra filen av prosess komponenten. Prosess komponenten

kaller så på metoder fra Input_handler hvor Input_handler kjører bildet fra video gjennom PyTorch rammeverket og YOLOv5 modellen. Deretter tegner resultater på bildet. Input_handler returnerer så det endelige resultatet tilbake til prosess komponenten.

Koden i figur 4.8 viser hvordan Input_handler sender bildet til YOLOv5 modellen og henter ut prediksjoner i form av koordinater og hvilken markering koordinatet hører til:

```
def predict_with_model(self, frame):

    self.model.to(self.device)
    frame = [frame]
    prediction = self.model(frame)

    labels, coords = prediction.xyxyn[0][:, -1],
                     prediction.xyxyn[0][:, :-1]

    return labels, coordinates
```

Figur 4.8: Funksjon som benytter YOLOv5 til å predikere på bilde. Returnerer koordinater for prediksjon samt hva som har blitt predikert.

Videre brukes koordinatene og markeringer som ble hentet ut fra metoden i figur 4.8 til å tegne avgrensningsbokser på bildet og hente ut konfidens verdier for prediksjonen. Verdien bestemmer om det skal telle som en deteksjon eller ikke. Figur 4.9 viser en forenklet *plot_frame* metode:

```
def plot_frame(self, prediction, frame, rawFrame):
    labels, coordinates = prediction
    labelLength = len(labels)
    x_shape, y_shape = frame.shape[1], frame.shape[0]

    for i in range(labelLength):
        row = coordinates[i]
        confidenceValue = row[4]

        if confidenceValue >= self.detectionThreshold:
            if self.captureDetection:
                lagreBilde()
            plotLabel()
            plotBox()

    return frame, detection_flag, detectionCount, lowestConfidence,
highestConfidence
```

Figur 4.9: Funksjon for å tegne avgrensningsbokser samt returnere verdier fra prediksjon.

4.2.4.4 Komponent: GUI

Ansvaret til komponenten består i å presentere og oppdatere det grafiske grensesnittet til å vise nåværende bilde av prosessert inndata og håndtere bruker-interaksjon på knapper.

Det grafiske grensesnittet er modellert ved bruk av Pythons innebygde tkinter bibliotek. (*Tkinter - Python interface to TCL/TK, u.å.*) Figur 4.10 viser hvordan det opprettes en instans av det grafiske grensesnittet:

```
class Gui_output:
    def __init__(self):

        self.root = tk.Tk()
        self.root.title(windowTitle)

        self.output_view = Gui_Setup(self.root)
        self.output_view.pack()
        self.output_view.update_source_title(sourceTitle)
```

Figur 4.10: Initiering av det grafiske grensesnittet

4.2.4.5 Komponent: MQTT_Subscriber

Ansvaret for komponenten ligger i å ta imot innkommende meldinger fra ekstern kilde. Melding inneholder en URL til en modell som ved mottak blir lastes ned og angis som ny standard modell for programsystemet.

```
newSource = None
def on_message(self, client, userdata, message):
      decodedMessage = dekod_mqtt_melding()
    try:
        sjekkGyldigMelding()
    except:
        Feilmelding
    if gyldigMelding and msg["newSource"] is not None:
        newSource = msg["newSource"]
    else:
        Feilmelding
```

Figur 4.11: Mottak av ny modell fra ekstern MQTT Publisher.

4.2.4.6 Komponent: MQTT_Publisher

Ansvaret for komponenten består i å sende ut melding som inneholder viktig data om nåværende status for programsystemet. Dataen består av tidspunkt, lokasjon til avsender samt informasjon om status på deteksjon, deriblant om det er en deteksjon, hvor mange dyr som er detektert og hvor sikker modellen er på at det er en deteksjon.

Meldingen blir sendt ved metoden *publishDefault* hvor prosess komponenten angir de forskjellige verdiene ut fra resultat av prosessering av bildet. For at dataene skal være enkelt å lese for både mennesker og maskin blir de konvertert til JSON-formatet.

```
def publishDefault(self, currentTime, currentLocation, detected_flag,
detectedCount, lowestConfidence, highestConfidence):
   msg = jsou.å.umps({'time' : currentTime,
                      'location' : currentLocation,
                      'detected' : detected_flag,
                      'detectedCount' : detectedCount,
                      'lowestConfidence' : lowestConfidence,
                      'highestConfidence' : highestConfidence
                      }, indent = 4)

   self.client.publish("DWLS_DETECTION", msg)
```

Figur 4.12: Publiserings funksjonen som konverterer verdier til JSON melding før de blir sendt.

5 RESULTATER

Prosjektet startet som et prosjekt hvor målsettingen var å lage en modell for deteksjon av hjortevilt. Prosjektet ble senere utvidet til å også inkludere et programsystem som tok inn video og benyttet modellen til å predikere og varsle ved deteksjon.

Som resultat for modellutvikling ble det produsert flere ulike modeller med ulike parametere. der noen versjoner av modellen hadde høyere presisjon, men ble mer krevende for maskinvare å kjøre, noe som medfører høyere prosesseringstid som vist i tabell 12 i kapittel 5.3.2.

5.1 Evalueringsmetode

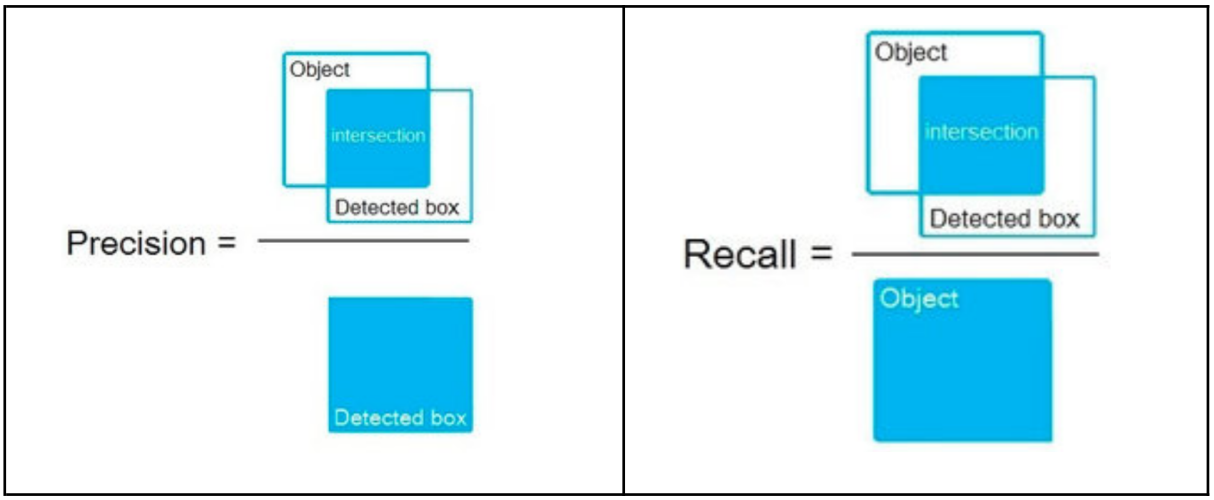
5.1.1 Evaluering av modell

Modellen evalueres på presisjon, recall og mAP.

Presisjon beregnes basert på antall objekter som er korrekt identifisert av totalt antall objekter. Et eksempel på utregning av presisjon kan være at modellen detekterer 10 hjort på et bilde hvor 9 av dem er korrekt, er presisjonen lik 90%.

Recall er antall korrekte objekt deteksjoner i forhold til antall faktiske objekt. I dette tilfellet, kan det være at modellen detekterer 5 hjort på et bilde hvor det egentlig er 10. Det vil føre til at recall er 50%.

Precision og recall brukes for å få en helhetlig, nøyaktig oversikt over ytelsen til modellen.



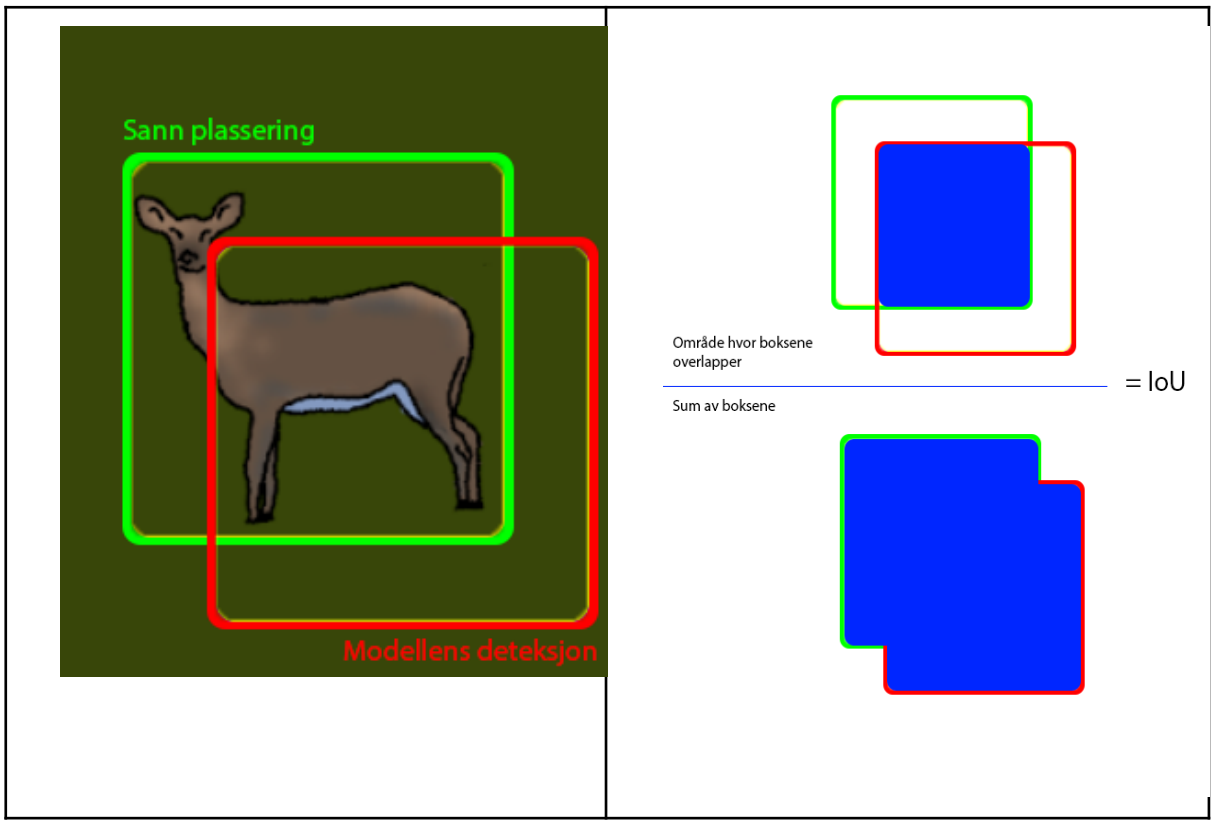
*Figur 5.1: Viser hvordan **precision** og **recall** blir utregnet. (Pershona, Zhanna. Kazdorf, Sergei, 2019)*

Modellen blir og testet på hvor godt den klarer å estimere posisjonen til et objekt. Det evalueres hvor mye overlapp det er mellom den estimerte posisjonen og den faktiske posisjonen, kjent som *Intersection over Union* (IoU) målt med mAP. Ved evaluering av modellen vil en få to verdier, mAP@0.5 og mAP@0.5:.95.

mAP@0.5 viser til treffsikkerheten på en gitt deteksjon hvor minimumskrav for IoU er satt til 50% dekning.

mAP*@0.5:.95* viser snittet beregnet på verdiene mellom 50%, 55%, … , 95%. Etter alle målingene er tatt, blir gjennomsnittet regnet ut slik at en får et mer nøyaktig bilde av hvor treffsikker modellen faktisk er.

Som mål vil en ha mAP på over 0.8 ved 50% dekke, og over 0.50 ved snitt av 50% til 95% dekke.



Figur 5.2: Visualisering for IoU. Det er lettere å gjøre deteksjon når kravet til overlapp er mindre.

5.1.2 Evaluering av programsystem

5.1.2.1 Systemtest

Programsystemet testes grundig med utgangspunkt i use casene (Se Vedlegg 3: Kravdokumentasjon) for hver enkelt metode som ble utviklet og lagt til. Under systemtest evalueres det at metodene gjør som forventet i forhold til use case og at eventuelle feil håndteres.

5.1.2.2 Brukertest

Det utføres en enkel brukertest hvor modell og programsystem testes på video levert av oppdragsgiver, hvor oppdragsgiver gir en vurdering.

For å sørge for at programsystemet tilfredsstiller oppdragsgivers krav gjennomføres en demonstrasjon og gjennomgang av programsystemet sammen med oppdragsgiver. Under gjennomgang demonstreres de implementerte funksjonene medfølgende en forklaring om motivasjon og mål for funksjonen.

5.1.2.3 Ytelsestest

For å vite at programsystemet oppfyller kravet om å prosessere data raskt er det nødvendig å måle programsystemets ytelse. For å evaluere at programsystemet oppfyller kravet måles gjennomsnittlig prosesseringstid over en periode på 5 minutter eller 5000 bilder, første mål som nås gjelder. Prosesseringstid per bilde lagres til en matrise og etter 5 minutter eller 5000 bilder er samlet beregnes snittet.

Målingen utføres for de forskjellige versjonene av modellen, ved bruk av GPU-akselerert prosessering, heretter henvist til som CUDA (*NVIDIA Developer, 2013*), eller kun CPU.

5.2 Evalueringsresultat

5.2.1 Modell

5.2.1.1 Evaluering av Precision, Recall og mAP

Tabell 5 viser resultater fra trening av de forskjellige YOLOv5 variantene. Forskjellene mellom de forskjellige variantene av YOLOv5 algoritmen består av antall parametere og nivåer som inngår i treningsprosessen.

Alle modellene blir trent på samme bildestørrelse. Den raskeste og minst krevende modellen Nano, har som antatt lavest nøyaktighet. Nøyaktighet stiger så for hver variant med unntak av X-Large hvor det forekommer en regresjon på recall, mAP@0.5 og mAP@0.5:.95.

Alle de forskjellige variantene i andre iterasjon har nådd målet om en mAP@0.5 > 0.8 og mAP@0.5:.95 > 0.5. Presisjon og recall har gode resultater på alle modellene i andre iterasjon.

Første iterasjon			Antall bilder: 185		
Modell	Bilde størrelse	Precision	Recall	mAP@ 0.5	mAP@ 0.5:.95
YOLOv5 Small	640x640	0.756	0.581	0.603	0.247

Tabell 5: Evalueringsresultat fra første iterasjon av YOLOv5 modellen.

Andre iterasjon			Antall bilder: 1736		
Modell	Bilde størrelse	Precision	Recall	mAP@ 0.5	mAP@ 0.5:.95
YOLOv5 Nano	640x640	0.893	0.837	0.869	0.544
YOLOv5 Small	640x640	0.897	0.865	0.898	0.564
YOLOv5 Medium	640x640	0.963	0.910	0.945	0.691
YOLOv5 Large	640x640	0.966	0.915	0.964	0.726
YOLOv5 X-Large	640x640	0.970	0.895	0.947	0.687

Tabell 6: Evaluerings resultat for de ulike versjonene av YOLOv5 modellen i andre iterasjon. Nano er den minste modellen, og X-Large er den største og mest krevende. Høyere verdier er bedre.

Sammenlignet med resultatene fra Kaarud, Nordvik og Paulsen (*Kaarud, Nordvik, og Paulsen, 2020)* er resultatene oppnådd her gode, hvor høyeste mAP@0.5 ligger på 96.64% her, mot deres 98.3%.

Merk at det er en forskjell på tilgjengelig treningssett i volum og kvalitet. Det er og i større grad er utfordrende og detektere hjortevilt med utgangspunkt i kontrast med hjorteviltets brune pels i kontra sauens hvite ull. Hjorten kan gå i ett med bakgrunnen og vises i andre vinkler sammenlignet med i deres studie hvor alle bilder besto utelukkende av liknende vinkler fra fugleperspektiv tatt av drone. Derfor er det vanskelig å ha en direkte sammenligning mellom resultatene.

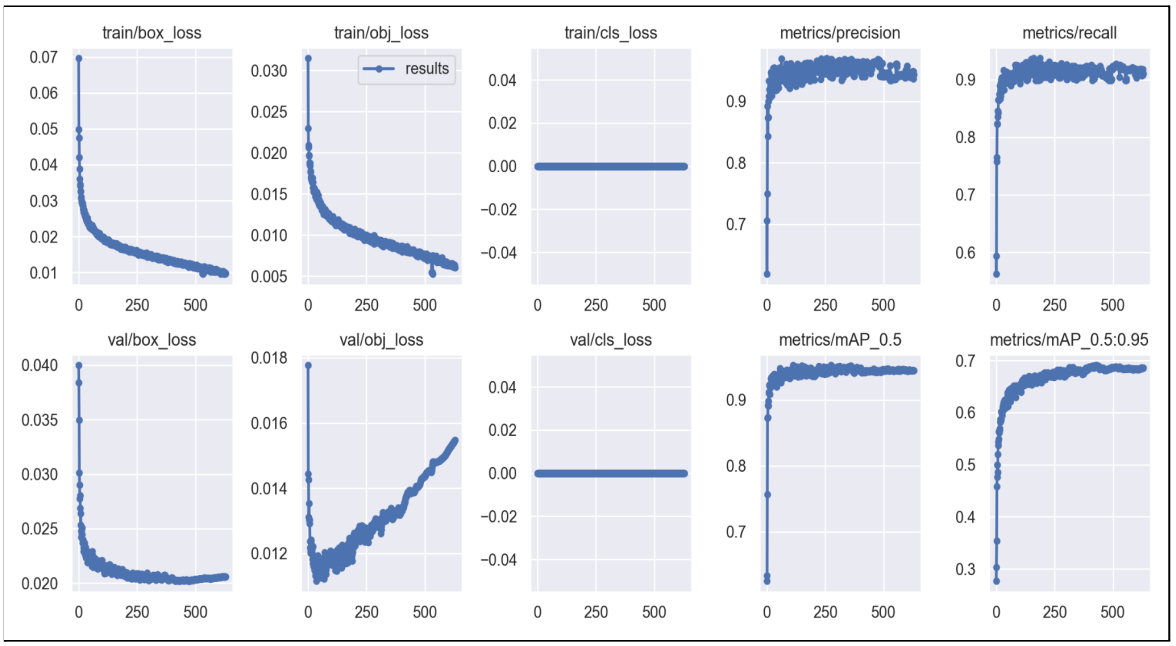
Figur 5.3 viser utvikling av verdier under trening av YOLOv5 Medium modellen, *Precision*, *mAP* og *Recall* er tidligere forklart i kapittel 5.1.1. Figuren er delt opp i statistikk fra treningssett(train) og valideringssett(val).

box_loss: Representerer hvor godt modellen identifiserer midtpunktet av et objekt og tegner avgrensningsboks rundt det sanne objektet.

obj_loss: Representerer målbarhet for sannsynlighet at objektet finnes i en foreslått region av bildet, hvis obj_loss er høy er sannsynlighet for at objektet finnes i bildet.

cls_loss: cls_loss gir en idé om hvor godt modellen kan predikere en korrekt klasse for et objekt. I dette prosjektet er det kun en klasse og cls_loss er derfor representert ved en strek uten variasjoner.

På graf val/box_loss og val/obj_loss øker verdiene etter noe som indikerer at valideringssettet overtilpasser treningssettet. Ideelt sett skal disse begynne å øke samtidig mens i dette tilfellet øker val/obj_loss på 30. iterasjon mens val/box_loss begynner å øke etter 500 iterasjoner.



Figur 5.3: Resultat fra en trening av YOLOv5 Medium

5.2.1.2 Brukertest

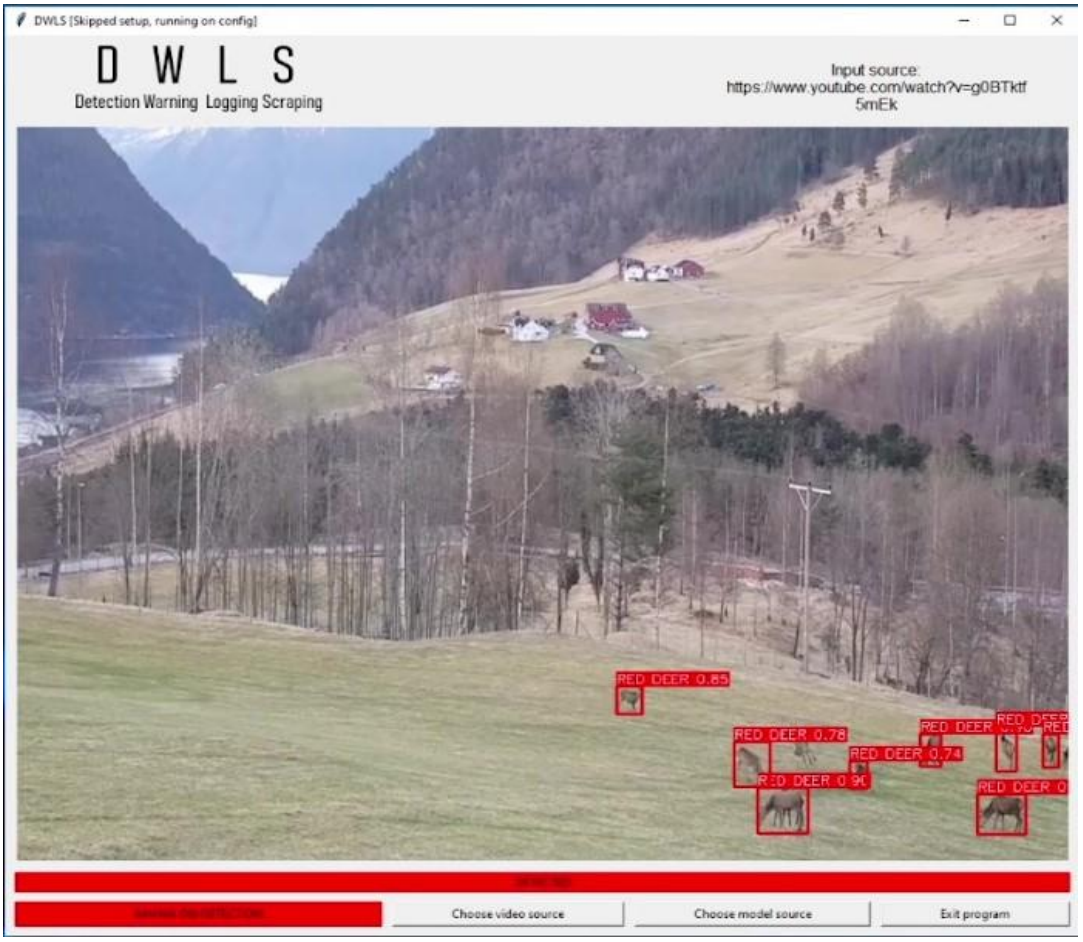
En demonstrasjon av programsystemet gjennomføres i kombinasjon med en begrenset brukertest hvor det leveres to forskjellige videoer av hjort, filmet av oppdragsgiver.

Videoene blir kjørt direkte fra YouTube. Programsystemet klarer å benytte YOLOv5-Medium modellen til å detektere hjorten nøyaktig og raskt nok. Gjennom mesteparten av videoen detekterer modellen med korrekt plassering på de fire hjortene som hovedsakelig var i bildet. Det samme gjelder for større grupperinger når inntil åtte hjort er i bildet som vist på figur 5.4.

Modellen klarer ikke detektere alle hjorter korrekt i alle tilfeller. I noen få sekunder når fire hjort var langt unna kamera, i et område der det var lav kontrast med bakgrunnen klarte ikke modellen å detektere hjortene nøyaktig. Modellen detekterer en hjort om gangen, men er ikke konsekvent på hvilken hjort.

Modellen detekterer i en situasjon taket til et hus som en hjort. Årsak til dette er at fargen på taket lignet på fargen på hjorten, i tillegg til at det var lav oppløsning på videoen.

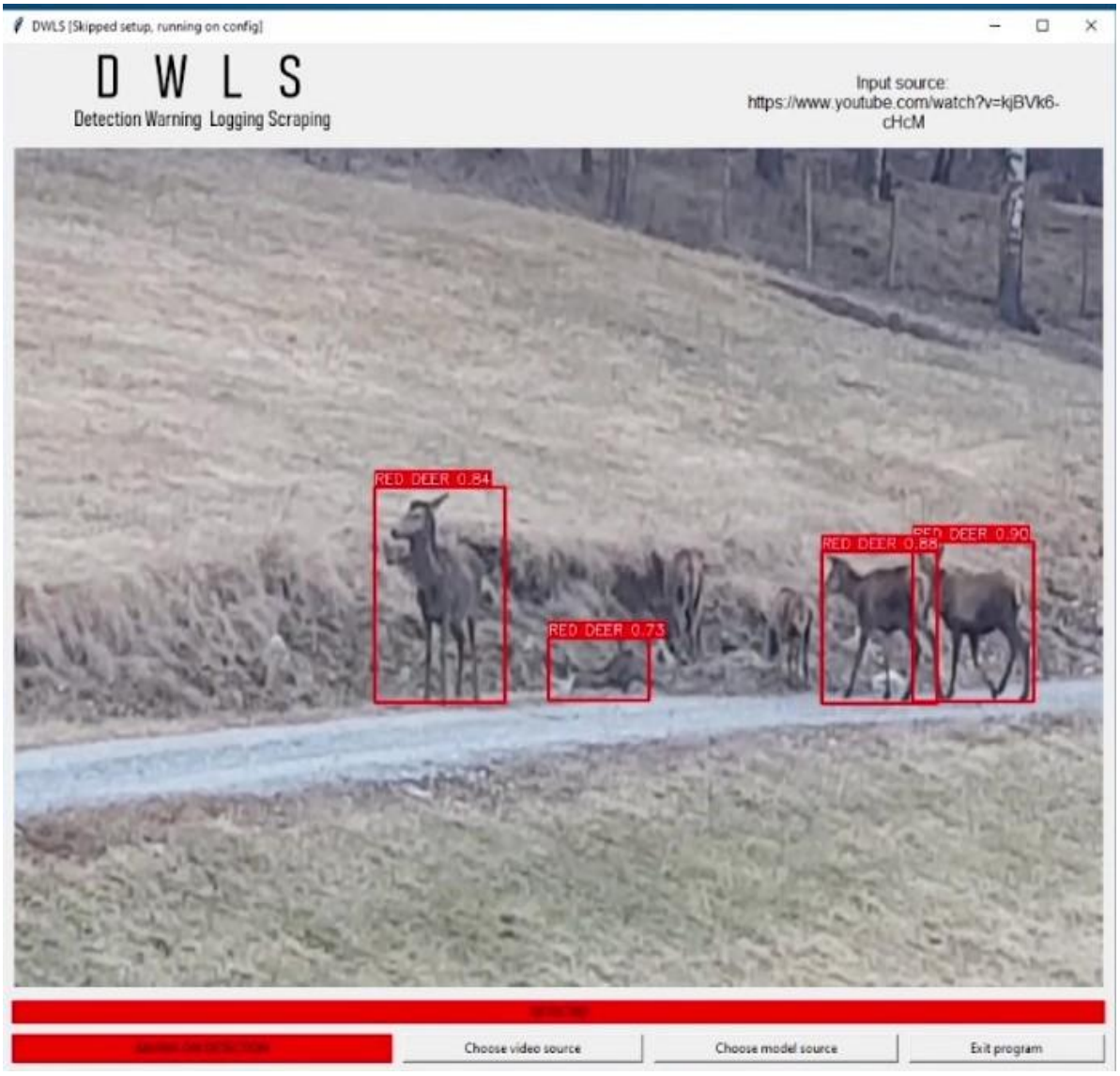
Ved stor bevegelse på hjorten er det vanskeligere for modellen for å følge detekteringen på noen av hjortene.



Figur 5.4: Skjermbilde fra videoen "Hjortesisten Vill Vest" som ble brukt under brukertest. (Hegland, 2021)

I video nummer to, også hentet fra YouTube blir programsystemet utfordret mer. Videoen viser en relativt tydelig en samling av hjort men hjorten står framfor en veikant hvor kontrasten var lav og fargen på bakgrunnen var lik fargen på hjorten selv.

Modellen har problemer med å detektere hjorten i øyeblikk hvor det er vanskelig å skille dyret fra bakgrunnen. Modellen klarer å detektere tre av de stående hjortene og til tider den ene hjorten som bader. De to stående hjortene i midten på figur 5.5 blir ikke detektert.



Figur 5.5: Bilde av den andre videoen "Hjortebading Vill Vest" brukt i brukertest. Merk at de to hjortene i midten ikke blir detektert. (Hegland, 2021)

5.2.2 Programsystem

5.2.2.1 Systemtest

Tester som gjennomføres, markeres med ✔ dersom testen fungerer som den skal og × dersom testen ikke fungerer som den skal.

Modul	Test	Resultat
ext_MQTT_publisher.py	Pushing av ekstern modell fra URL	✔
ext_MQTT_subscriber.py	Korrekt loggføring av meldinger	✔
	Korrekt validering av meldinger	✔
	Korrekt basert på inndata fra meldinger	✔
gui_output.py	Korrekt oppdatering av update_alarm_status()	✔
	Korrekt oppdatering av update_savingDetection_status()	✔
	Korrekt oppdatering av update_title_from_input_source()	✔
input_handler.py	Korrekt oppdatering av update_alarm_status()	✔
	Korrekt bruk av CUDA eller CPU	✔
	Korrekt innlastning av riktig modell	✔
	Korrekt prediksjon basert på brukerdefinert grenseverdi for konfidens	✔
	Korrekt lagring av bilder i forhold til intervall og håndtering av duplikater	✔
	Korrekt endring av størrelse på bilde	✔
MQTT_Subscriber.py	Korrekt validering av meldinger	✔
process.py	Korrekt håndtering dersom ingen inn-data	✔
	Korrekt håndtering ved valg av ny kilde	✔
	Korrekt håndtering dersom programmet kjører uten grafisk grensesnitt	✔
startup_setup.py	Korrekt return type og verdi	✔
	Korrekt håndtering av duplikat fil ved nedlastning av ny modell	✔
main.py	Korrekt håndtering av innstillinger definert i config.py	✔
	Korrekt definering av bilder per sekund fra inndata	✔
	Korrekt oppsett ved valg av startoppsett	✔

Tabell 7: I alle systemtester er resultatet at testen oppfører seg som forventet.

5.2.2.2 Brukertest

Ved demonstrasjon av programsystemet gjennomføres en begrenset brukertest av modellen og programsystemet samlet, nærmere forklart under *5.2.1.3 Brukertest.*

Tilbakemelding fra oppdragsgiver er at programsystemet fungerer veldig godt utover kravene som er satt, da tilleggsfunksjoner er praktiske. Det er positivt at det er tatt hensyn til vedlikeholdbarhet og ytelse ved kjøring på enheter med svakere maskinvare. Oppsummert er oppdragsgiver fornøyd og konkluderte med at krav var oppfylt og at potensial og utvikling videre er lovende.

5.2.2.3 Ytelsesvurdering

Testene blir utført på en stasjonær PC med maskinvare og parametere:

- CPU: AMD Ryzen 5800X
- GPU: GeForce RTX 3080Ti 12gb
- .m4v videofil med naturlig oppløsning 640x640
- Programsystemet kjører uten grafisk grensesnitt

Parametere for videofil og programsystemet blir valgt for å etterligne test omgivelsene til resultatene fra YOLOv5 ytelsestest utført av Qengineering. (Qengineering, u.å.)

Da en Raspberry Pi 4 (*Raspberry Pi Foundation, 2019)* ikke er tilgjengelig for testing blir ytelse estimert ut fra resultater på testmaskin omregnet til verdier for Raspberry Pi 4 og Jetson Nano.

Referanse resultatet er oppnådd med YOLOv5-Small, bildestørrelse 640x640 på en Raspberry Pi 4 64 bit 1950Mhz.

Referanse resultater og estimert overføringsverdi for ytelse:

Raspberry Pi 4	*1.6 FPS, tid per bilde 625 ms* *Testsystemet er 686% raskere.*	**Jetson Nano**	*4.0 FPS, tid per bilde 250 ms* *Testsystemet er 275% raskere*

Tabell 8: Referanse resultat for Raspberry Pi 4 og estimert overføringsverdi.

	Prosesseringstid (ms)		Estimert prosesseringstid (ms) og bilder per sekund(FPS)			
	CUDA	CPU	Raspberry Pi 4		Jetson Nano	
YOLOv5 Nano	10.7 ms	42.9 ms	294 ms	3.4 FPS	118 ms	8.5 FPS
YOLOv5 Small	10.5 ms	91.4 ms	627 ms	1.6 FPS	251 ms	4.0 FPS
YOLOv5 Medium	13.0 ms	217.5 ms	1492 ms	0.7 FPS	598 ms	1.7 FPS
YOLOv5 Large	15.5 ms	379.6 ms	2604 ms	0.4 FPS	1044 ms	0.9 FPS
YOLOv5 X-Large	24.1 ms	654.8 ms	4492 ms	0.2 FPS	1800 ms	0.5 FPS

Tabell 9: Resultat av ytelsestest og estimert ytelse på Raspberry Pi 4 og Jetson Nano

I tabell 9 viser ytelse på testmaskin samt estimert ytelse målt i prosesseringstid per bilde målt i millisekund (ms) og verdi konvertert til bilder per sekund (FPS). Generelt prosesserer Jetson Nano bilder raskere enn Raspberry Pi 4 da Jetson Nano kommer har integrerte CUDA-kjerner som gjør det mulig å prosessere data raskere enn på kun prosessor (*Jetson Nano Developer Kit,* 2021).

I forhold til krav om prosesseringstid på mindre enn 2000 ms vil følgende modeller gi best ytelse per enhet:

Raspberry Pi 4: YOLOv5 Small *627 ms per bilde*
YOLOv5 Medium *1492 ms per bilde*

Jetson Nano: YOLOv5 Medium *598 ms per bilde*
YOLOv5 Large *1044 ms per bilde*

5.3 Prosjektresultat

5.3.1 Funksjonelle og ikke funksjonelle krav

Prosjektets krav er delt opp i funksjonelle og ikke-funksjonelle krav (se Vedlegg 1: Visjonsdokument). I løpet av prosjektet ble noen eller deler av krav fjernet etter avgrensning av prosjekt.

Tabellene under viser de funksjonelle- og ikke-funksjonelle krav som ble implementert i prosjektet. Det vises og hvilke delkrav disse består av for å oppfylle det funksjonelle kravet. I tabellene er planlagte delkrav markert med ⚪, bortprioriterte krav ⫽, fullført ✔ ikke fullført ×.

Tabell 10: Implementerte funksjonelle krav

Nr.	Funksjonelt krav	Delkrav			
		Nr.	Beskrivelse	Planlagt	Fullført
1	*Produktet må kunne ta inn bilder eller video.*	1.1	Programsystem tar inn data i form av video	⚪	✔
		1.2	Programsystem tar inn data i form av bilde	⫽	×
		1.3	Konvertere video til bilder	⚪	✔
2	*Produktet må kunne monitoreres, for å vurdere ytelse og nøyaktighet*	2.1	Måling av ytelse for programsystem	⚪	✔
		2.2	Måling av nøyaktighet	⚪	✔
3	*Produktet må kunne trenes opp på flere dyreslag slik at det kan bli implementert der det er nødvendig/aktuelt.*	3.1	Modell kan trenes opp på andre arter	⚪	✔
		3.2	Programsystem lagrer bilder ved deteksjon	⚪	✔
4	*Programsystem må kunne motta visuell inndata fra kamera/sensor.*	4.1	Inndata fra lokalt kamera	⚪	✔
		4.2	Inndata fra URL	⚪	✔
		4.3	Inndata fra lokal mediefil	⚪	✔

Nr.	Funksjonelt krav	Delkrav			
		Nr.	Beskrivelse	Planlagt	Fullført
5	*Produktet må kunne prosessere og analysere bilder for hjortevilt.*	5.1	Prosessering av hjort	⚪	✔
		5.2	Prosessering av elg	⫽	×
		5.3	Prosessering av rein	⫽	×
		5.4	Prosessering av rådyr	⫽	×
6	*Produktet må kunne varsle om hjortevilt.*	6.1	Detektere om der er dyr i nåværende bilde	⚪	✔
		6.2	Samle data fra deteksjon til varsling	⚪	✔
		6.3	Sende varsling	⚪	✔
		6.4	Vise varsling	⚪	✔

Tabell 10: Implementerte funksjonelle krav

Nr.	Beskrivelse:	Planlagt	Fullført
1	**Hastighet**: Produktet må kunne prosessere inndata raskt nok til at systemet kan sende ut varsel i løpet av få (0.5-2) sekunder	⚪	✔
2	**Pålitelighet**: Produktet må minimere falske negative eller falske positive varslinger.	⚪	✔
3	**Tilgjengelighet**: Det skal være lett å distribuere og installere på en gitt lokasjon.	⚪	✔
4	**Vedlikehold**: Det skal være relativt lett å vedlikeholde eller fikse dersom det skulle oppstå feil på programsystem eller modell.	⚪	✔
5	**Fleksibilitet:** Produktet må kunne skille mellom f.eks hjortevilt, rådyr eller elg under ulike forhold som kan oppstå i løpet av årstidene.	⫽	×

Tabell 11: Implementerte ikke-funksjonelle krav

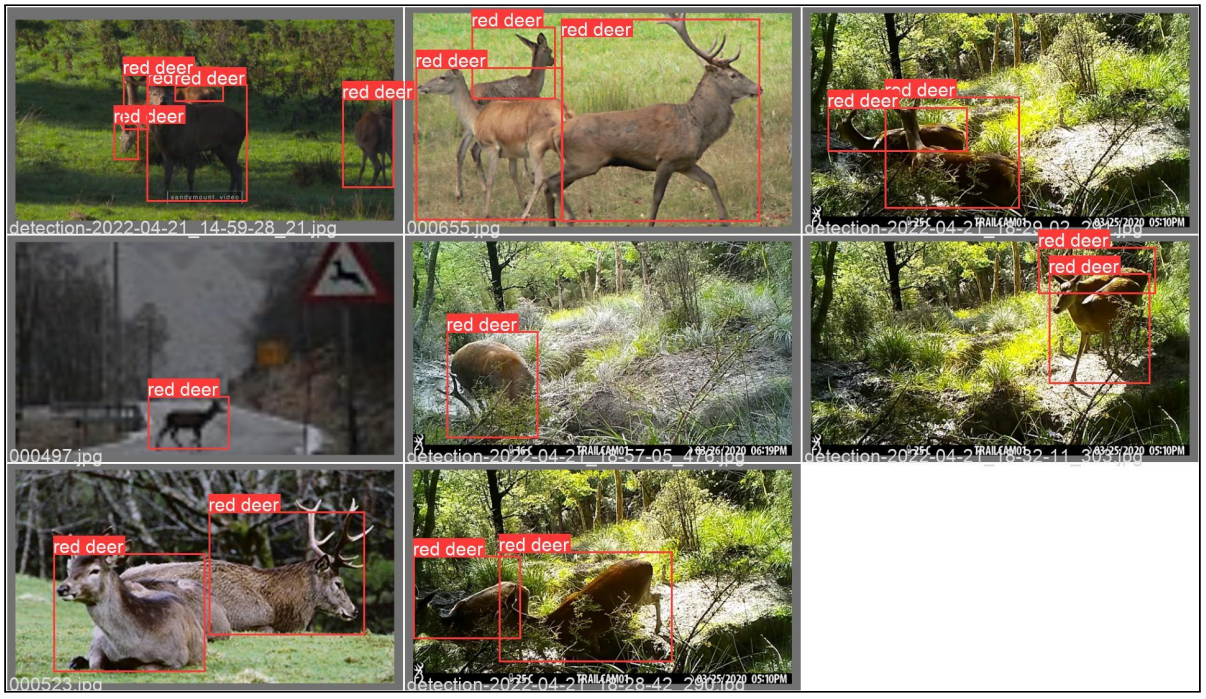
5.3.2 Modell

Modell	Precision	Recall	mAP@ 0.5	mAP@ 0.5:.95	Estimert prosesseringstid (ms) og bilder per sekund(FPS)			
					Raspberry Pi 4		Jetson Nano	
YOLOv5 Small	0.897	0.865	0.898	0.564	627 ms	1.6 FPS	251 ms	4.0 FPS
YOLOv5 Medium	0.963	0.910	0.945	0.691	1492 ms	0.7 FPS	598 ms	1.7 FPS
YOLOv5 Large	0.966	0.915	0.964	0.726	×	×	1044 ms	0.9 FPS

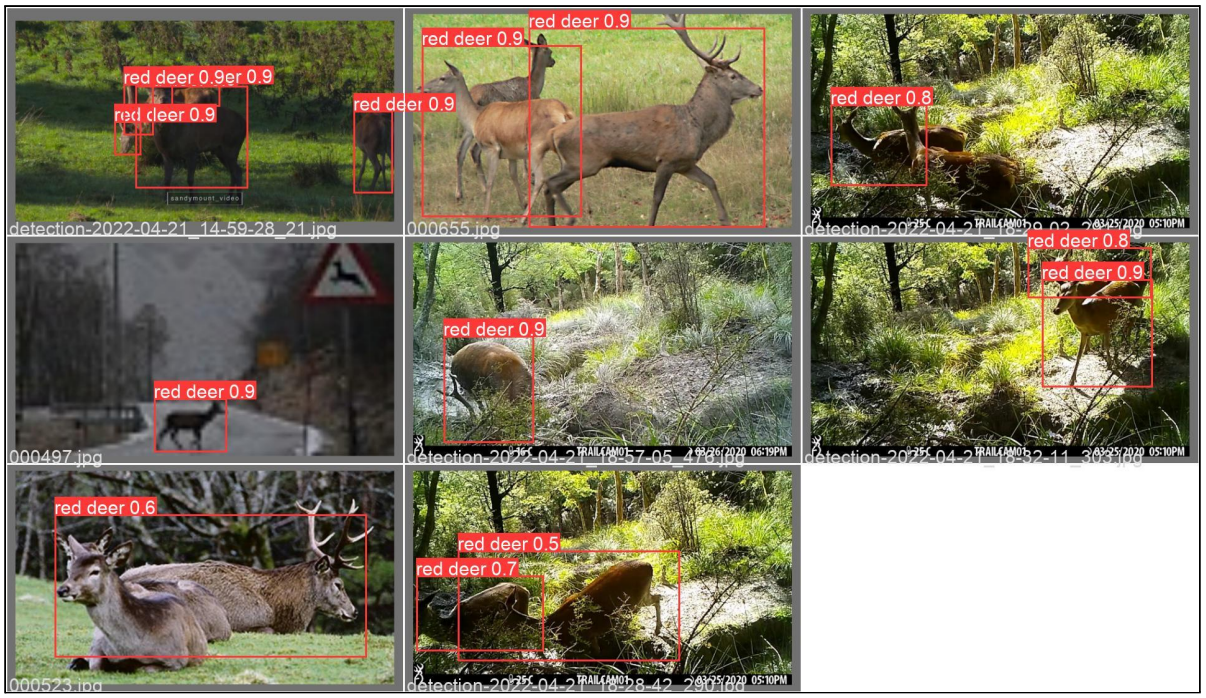
Tabell 12: Utvalgte YOLOv5 versjoner til bruk på Raspberry Pi 4 og Jetson Nano

Figur 5.6 og 5.7 under viser resultatet av en kjøring av modellen mot valideringssett. Modellen klarer med stor nøyaktighet å detektere hjort i bildet med noen unntak.

På figur 5.7 bilde 1-3 og 3-1 detekterer modellen to individer som et enkelt individ og klarer ikke skille de to individene.

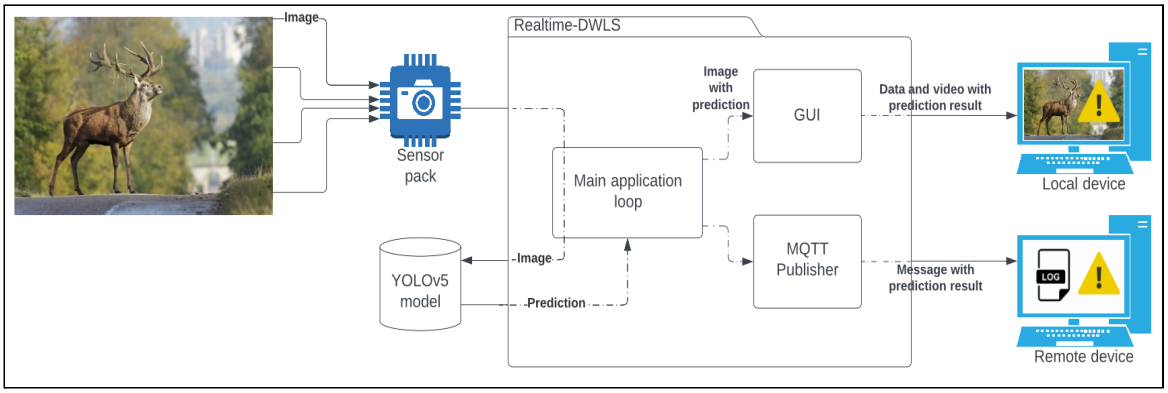


Figur 5.6: Eksempel samling av bilder fra valideringssett.



Figur 5.7: Modellens prediksjon på valideringssettet, merk at på bilde 1-3 og 3-1 og blir to hjorter identifisert som et individ.

5.3.3 Programsystem

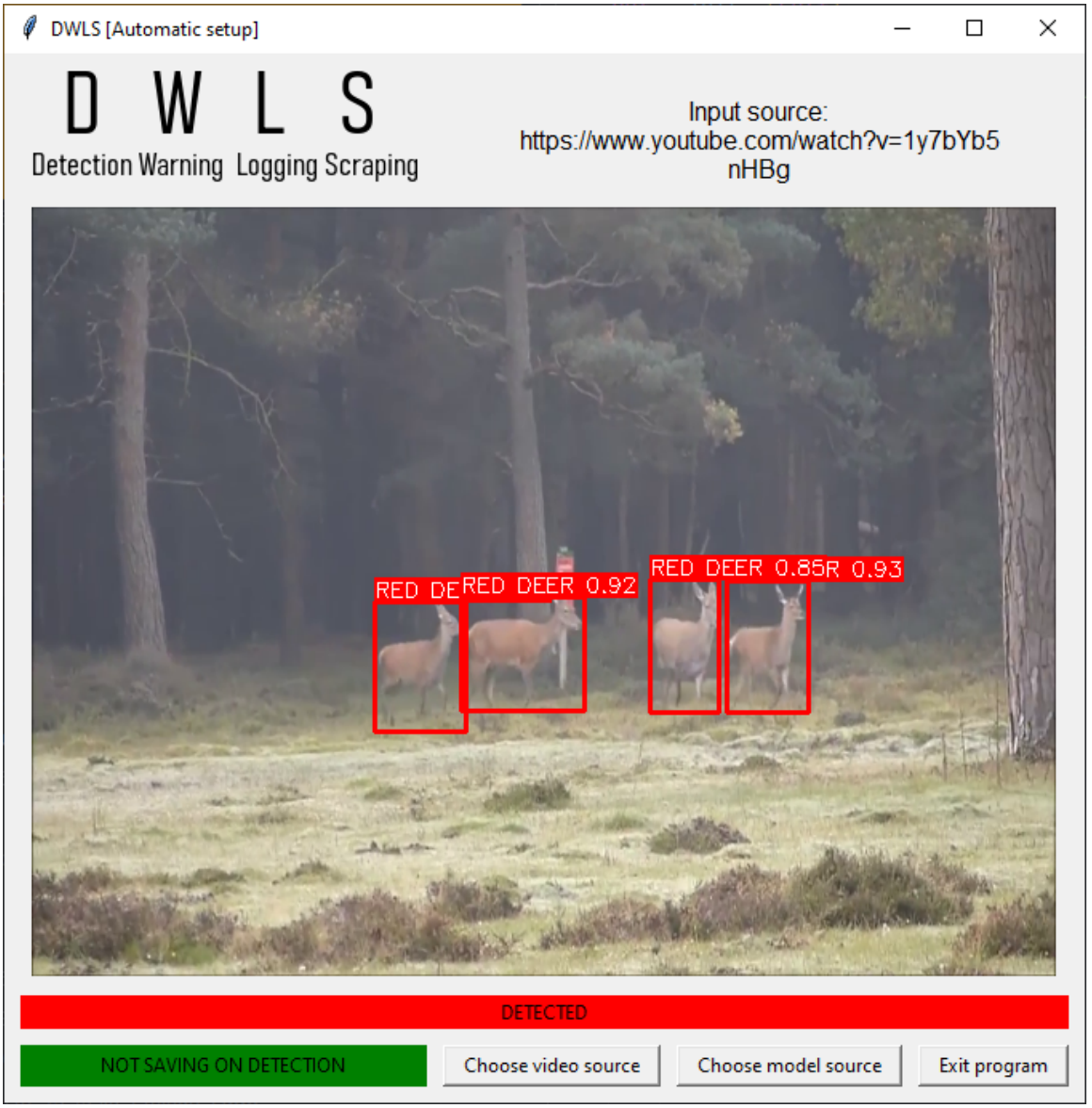


Figur 5.8: Arkitekturskisse av utviklet programsystem

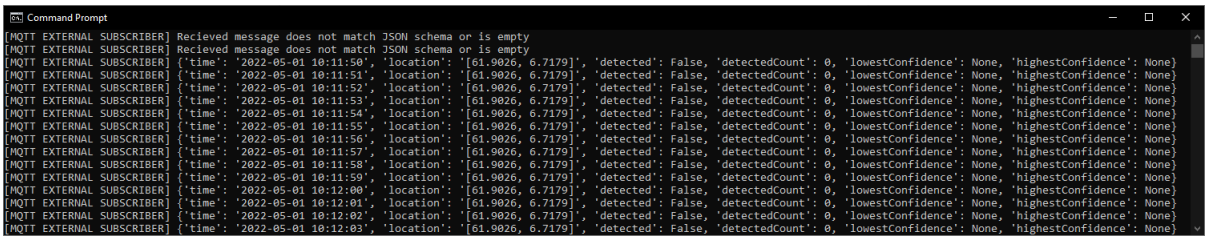
Figur 5.8 over viser en overordnet flyt for programsystemet i *use case: Deteksjon*. Sensor systemet tar bilde og sender til programsystemet. Programsystemet sender bildet gjennom modellen for prediksjon og returnerer verdier som inneholder informasjon om prediksjonen, samt et bilde med opptegnede avgrensningsbokser.

Bildet sendes så til grafisk grensesnitt for visning, informasjon relatert til prediksjonen blir sendt ut i en melding via MQTT til en ekstern enhet for loggføring. Varsling blir iverksatt på den eksterne enheten og i den lokale enheten dersom det grafiske grensesnittet er i bruk.

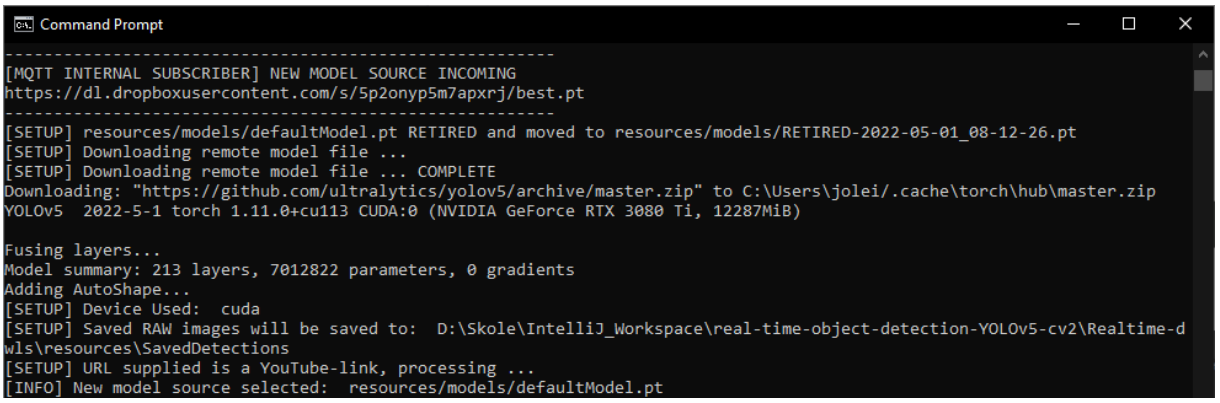
Figur 5.9 under viser det grafiske grensesnittet under kjøring av programmet. Her detekteres fire hjort med hver sin avgrensningsboks og konfidensverdi. Under bildet viser varslingsindikator at det er blitt detektert hjort i bildet og lagringsindikator viser at bilder av deteksjoner ikke blir lagret. Det grafiske grensesnittet har og tre knapper med funksjon for valg av video- og modell kilde, samt en knapp for å avslutte programmet.



Figur 5.9: Programsystemets hovedvindu i det grafiske grensesnittet



Figur 5.10: Utskrift i ledetekst på hva som blir mottatt av en ekstern mottaker, innholdet i meldingen blir loggført.



Figur 5.11: Utskrift fra ledeteksten i det hovedprogrammet mottar en ny modell fra en ekstern enhet.

Figuren 5.10 viser utskrift i ledetekst av meldinger mottatt av ekstern MQTT Subscriber. Meldingene inneholder tidspunkt, lokasjon, status på deteksjon, hvor mange deteksjoner som blir gjort i bildet samt høyeste og laveste konfidensverdi.

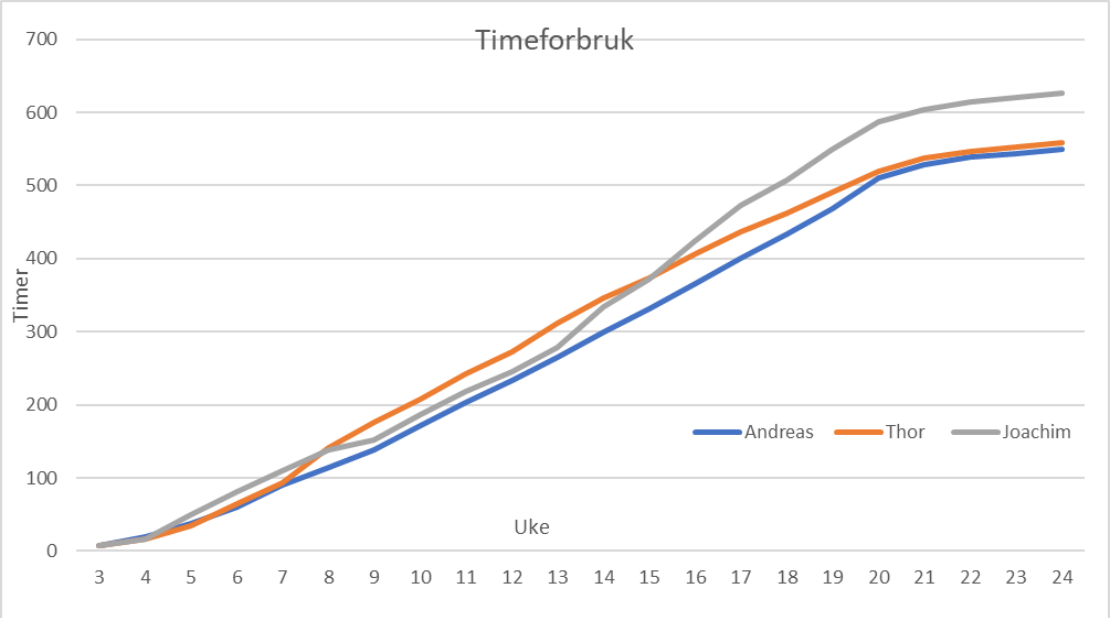
Figur 5.11 viser ledetekst til programsystemet i det det mottar en melding fra en ekstern MQTT Publisher. Meldingen inneholder en URL til en ny *oppdatert* modell som skal benyttes i programsystemet. Programsystemet laster så ned den nye modellen og setter til side den daværende modellen. Den nye modellen blir så satt som ny standard modell og tas i bruk.

5.4 Prosjektgjennomføring

Prosjektet har i stor grad klart å følge framdriftsplan framstilt i figur 3.4.
Prosjektets framdrift har vært jevn gjennom hele med økende tidsbruk mot siste halvdel av prosjektets tidsramme med unntak av noen små perioder hvor tidsbruk har falt litt grunnet arbeid med andre fag forfatterne deltok i samtidig med prosjektets gjennomføring. Forefallende arbeid ble hentet inn i løpet av påfølgende tidsperiode (Se Vedlegg 2, Prosjekthåndbok: Statusrapporter).

Generell tidsbruk lå i prosjektets første halvdel litt under forventet tidsbruk per periode, problemet ble identifisert og det ble iverksatt tiltak for å fullføre oppgaver i forhold til tids- og framdriftsplan. Tidsbruk i siste halvdel av prosjektet økte markant og holdt seg stabil til prosjektets fullføring som vist i figur 5.12. Den totale tidsbruken for prosjektet endte på 1734 timer og overskred tidsbudsjettet på 1620 timer med 114 timer.

Alle planlagte aktiviteter ble gjennomført i løpet av prosjektet og alle leveranser ble oppfylt.



Figur 5.12: Viser det sammenlagte timeforbruket til hver person gjennom prosjektet.

6 DISKUSJON

I dette kapittelet diskuteres gjennomføring og resultater. Det drøftes hva som gikk bra og hva som viste seg å være utfordrende med potensielle løsninger på disse problemstillingene.

6.1 Valg av algoritme

En egenutviklet algoritme ble vurdert, men ble valgt bort da det var usikkert om resultatet faktisk ville bli bedre enn de allerede eksisterende algoritmene. Det var uvisst hvor lang tid det ville ta å utvikle og risiko for å ikke rekke fullføre prosjektet var for stor.

You Only Look Once (YOLO), *Single Shot Detector* (SSD) og *Faster Region based Convolutional Neural Network* (FRCNN) ble vurdert for prosjektet.

SSD fungerer på samme måte som YOLO velges bort da den krever bilder med høy oppløsning, noe prosjektet ikke hadde tilgjengelig.

FRCNN ble vurdert da den hadde høyest treffsikkerhet. Da YOLO har en tyngre variant som kan gi mer eller mindre like resultat var det naturlig å velge YOLO da den har den flere varianter som kan brukes ved forskjellige applikasjoner og maskinvare noe som gir fleksibilitet i forhold til nøyaktighet og krav til maskinvare.

YOLO algoritmen ble tidlig vurdert som en potensiell løsning på den initielle problemstillingen. Small varianten av YOLO ble testet tidlig i første iterasjon på et svært lite datasett med kun 185 bilder. Modellen presterte bra til tross for størrelsen på datasettet, men modellen gav en del falske positive og negative i situasjoner med tynne trær og busker som kunne ligne på gevir.

I andre iterasjon av innsamling av bilder økte datasettet til 1736 bilder og et grovt estimat på to hjort per bilde, altså ~3-4000 instanser av hjort. Datasettet var for lite i forhold til anbefalt størrelse hvor det anbefales at datasettet minimum burde inneholde 1500 bilder og 10.000 instanser av objektet som skal detekteres (Ultralytics, 2020).

Dersom prosjektet skulle blitt gjennomført på nytt bør det gjøres en grundigere sammenligning mellom forskjellige algoritmer trent på samme datasett og kjørt på faktisk maskinvare for å måle ytelse i forhold til nøyaktighet. Det burde og ha blitt lagt mer vekt på å samle inn data fra bruksområde eller liknende brukstilfeller for i større grad kunne trene en modell og evaluere den i situasjoner hvor den er tiltenkt benyttet.

6.2 Modellutvikling

Det viste seg at beslutning om å avgrense datasettet til å kun fokusere på hjort i normale fargebilder gjort tidlig i prosjektet var en god avgjørelse. Det viste seg utfordrende å finne passende bildemateriale for brukstilfelle. Tilgjengelige bilder består hovedsakelig av høyoppløselige naturfotografier, gjerne nærbilder som ikke er direkte overførbart til prosjektets bruksområde hvor hjort ofte er okkludert eller vises fra hvilken som helst vinkel, ofte over en lengre distanse.

Dette og maskinvareprosjektet hadde ulike fremdriftsplaner og frister, det ville bli vanskelig å organisere produktiv samhandling mellom prosjektene. Det ble derfor gjort en avgrensning om at prosjektene kjørte selvstendige løp uavhengig av hverandre.

Brukertestene viste svakheter ved modellen, deteksjon av hjort i bevegelse ga lavere konfidensverdier og falske negative varslinger. En mulig årsak til dette er at den ikke har tilstrekkelig data i treningssettet og ikke har blitt eksponert nok til hjorter i bevegelse. En annen mulig årsak er at kameraet ved rask bevegelse mister fokus eller viser uklarheter på bildet som gjør det utfordrende for modellen å kjenne igjen objektet.

Modellen slet og med å detektere hjort ved stor avstand fra kamera da objektene blir for små til å ha karakteristiske trekk.

Det var og utfordrende for modellen å kjenne igjen objekter i bilder hvor kontrasten var lav som vist i figur 5.5. Det kom og fram i testene at brun eller mørk bakgrunn skapte problematikk da det i noen grad kan ligne på pelsen til dyrene.

For å håndtere disse problemstillingene ville det vært ideelt med kamera som klarer følge små objekter i bevegelse med skarpt bilde, eller at kontrasten på bildene økes under bildebehandling og prosessering. En annen mulighet er å benytte infrarødt eller termisk kamera som gir bilder i høy kontrast. Dette forutsetter at modellen blir trent opp på et datasett hvor bilder fra nevnte kamera inngår.

Om prosjektet skulle gjennomføres på nytt kunne resultatet blitt forbedret ved økt samhandling med maskinvareprosjektet. Det ville i større grad vært mulig å spesialisere modell mot tilgjengelige sensorer. Det ville og vært mulighet for å teste reell ytelse på faktisk maskinvare. Det kunne og vært en mulighet å plassere en prototype-løsning ut i felt for å forsøke hente inn supplerende data fra faktisk bruksområde.

6.3 Utvikling av programsystem

Prototypen av programsystemet ble en praktisk måte å demonstrere hvordan programsystemet kunne implementeres med sensorsystemet til et felles varslingssystem.

Funksjoner i programsystemet bidro til å gjøre det enkelt kunne samle inn større mengder med data til opptrening av ny modell. Samtidig bidro programsystemet til å gjøre det enklere å foreta en visuell evaluering av modellens ytelse på test video.

Programsystemet oppfylte krav som ble satt. Resultatet av brukertest hvor modellens ytelse, presisjon og de ulike funksjonene implementert i programsystemet ble demonstrert var at oppdragsgiver var godt fornøyd og påpekte at løsningene var gode og resultatet virket lovende for videre arbeid.

Utfordringer ved programsystemet ligger hovedsakelig i arkitekturen hvor programsystemet består av en større enhet som tar inn data og prosesserer dette. Dette kunne vært delt opp til mindre individuelle enheter som kunne kommunisert med hverandre via nettverk. Dette hadde lagt til rette for at programsystemet ville vært mer fleksibelt og modulært. Det kunne da lettere blitt tilpasset forskjellige bruksområder.

Samtidig er det en begrensning i prototypen som ikke kan håndtere mer enn en datakilde om gangen. I praksis vil et sensorsystem kunne bestå av flere kameraer som peker i forskjellige retninger.

Dersom programsystemet skulle blitt utviklet på nytt bør det vurderes å utvikle programsystemet som en web-applikasjon hvor de individuelle modulene kan flyttes på mellom forskjellige enheter for å gjøre løsningen mer fleksibel. Det hadde og vært nyttig å legge mer vekt på enhetstesting og utformings prinsipper for å gjøre programsystemet mer robust og vedlikeholdbart.

6.5 Fremgangsmåte

Tidlig i prosjektet ble der gjort flere avgrensninger grunnet var lav tilgang på treningsdata fra ulike typer kamera. Fokus på dette ville medføre mindre tid til utvikling og trening av modell uten garanti for at datasettet hadde blitt stort og godt nok. Prosjektet ble derfor avgrenset til kun basere seg på fargebilder av hjort.

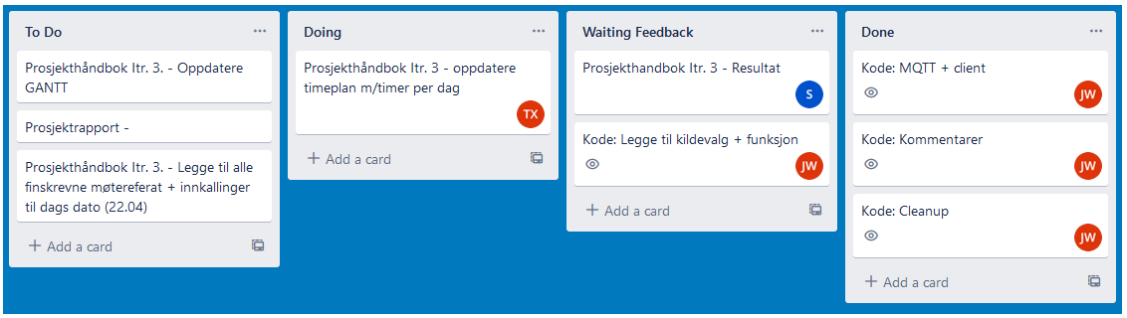
Dette prosjektet var et delprosjekt av et større todelt prosjekt der det andre delprosjektet skulle stå for utviklingen av maskinvare, det viste seg tidlig i prosjektutviklingen at delprosjektene ikke ville få klargjort en felles prototype til testing i felt innenfor tidsrammen hvor hjortevilt var lett å fotografere. Dette førte til førte til at testing måtte simuleres med hjelp av videoer og bilder tilsendt av oppdragsgiver.

I etterkant av prosjektet har det vist seg at en høyere frekvens på korrespondanse med oppdragsgiver og maskinvareprosjektet kunne hjulpet på sluttresultatet på prosjektet.

Tettere møter med utveksling av framdriftsrapporter kunne lagt til rette for at delprosjektene kunne produsert en felles prototype som hadde hjulpet til med testing, måling av ytelse og innsamling av data.

6.6 Prosjektgjennomføring

Gjennomføringen av prosjektet har fungert godt. Det har blitt jobbet jevnt med krav, noe som hindret forsinkelser og perioder med store arbeidsmengder. Prosjektet hadde hyppige møter innad og benyttet verktøyet Trello for å organisere arbeidsoppgaver som vist i figur 6.1. Det ble organisert jevne møter med veileder hvor status på fremdrift, krav og problemstillinger ble drøftet.



Figur 6.1: Viser et utdrag fra Kanban board gjennom tjenesten Trello(Trello, u.å.)

Prosjektplanen har stemt godt overens med den iterative prosessen. Prosjektets omfang og krav ble tydelig definert tidlig i prosjektet og har blitt videreutviklet deretter med noen endringer underveis.

En prototype av modell ble tidlig klar i utviklingsprosessen og det ble stilt spørsmål rundt testing og evaluering av denne. På grunnlag av dette og tilgjengelig kapasitet som følge av avgrensningene ble prosjektet utvidet til å også omfatte et programsystem som kunne ta inn video og benytte modellen til å utføre prediksjon og varsling.

Hensikten med programsystemet var å lage et brukergrensesnitt som gjør modellen lettere å ta i bruk, demonstrerer hvordan modellen kan inngå i et system og gir muligheten for brukertesting.

Selv om det var positivt at der ble utviklet programvare til modellen, burde utvikling av programsystemet startet tidligere. Mye tid i siste halvdel av prosjektet ble brukt på å lære programmeringsspråket Python og de forskjellige rammeverkene brukt i programsystemet. Dette førte til at prototypen av programsystemet ble ferdigstilt relativt sent i prosjektets løp.

Prosjektet overskred tidsbudsjettet med 6.57%. Arbeidsmengden viste seg å være stor og noen deloppgaver krevde litt mer tid enn først forventet. Det ble lagt vekt på god dokumentasjon, prosjektet hadde hyppige møter med veileder hvor tilbakemeldinger resulterte ble tatt hensyn til og det ble gjennomført korrektur i dokumentasjon. Overskridelsen i tidsbudsjettet kan være grunnet implementering av ekstra funksjoner i programsystemet eller høy fokus på korrekt utforming programsystem og dokumentasjon.

7 KONKLUSJON OG VIDERE ARBEID

Dette kapittelet går over konklusjon for prosjektets prosesser, resultater og anbefaling for videre utvikling av både modell og programsystem.

7.1 Konklusjon av prosjekt

Målet for prosjektet er å lage en dyplæringsmodell som detektere hjort, nærmere beskrevet i kapittel 1.4. Tidlig i utviklingsprosessen ble en tidlig versjon modellen produsert. I samråd med oppdragsgiver og veileder ble prosjektet utvidet til også å omfatte et programsystem som skulle handtere modellen og varsle ved deteksjon.

I løpet av prosjektperioden, har det blitt utviklet flere variasjoner av dyplæringsmodellen YOLOv5 og et støttende programsystem.

Ved ferdigstilling av prosjekt ble flere ferdig trente dyplæringsmodeller og et programsystem for varsling og anvendelse av modellene overlevert. De forskjellige variantene av modellene hadde forskjell i mAP på 89.3-97.0% som vist i tabell 6 i kapittel 5.1.1. I de forskjellige variantene er raskere prosesseringstid forbundet med lavere presisjon og lavere krav til maskinvare.

- For Raspberry Pi 4 anbefales det å benytte YOLOv5-Small som gir 1.6 bilder i sekundet med mAP 0.898 som vist i kapittel 5.3.2: Tabell 12.
- For Jetson Nano anbefales det å benytte YOLOv5-Medium som gir 1.7 bilder i sekundet med mAP 0.945 eller YOLOv5-Large som gir 0.9 bilder i sekundet med mAP 0.964 som vist i kapittel 5.3.2: Tabell 12.

Programsystemet ble levert med flere funksjoner for lettere drift og vedlikehold, som funksjon for bytte av modell, funksjon for bytte av kamera og funksjoner for lokal eller varsling over nett som vist i kapittel 5.3.3: Figur 5.8.

Målene for prosjektet ble nådd med gode resultater tross avgrensninger og utfordringer i innsamling av nok data til trening av modellen. Programsystemet var en utvidelse av det originale målet for prosjektet og bidro til enklere evaluering og demonstrasjon av både modell og hvordan et eventuelt programsystem kan benyttes i brukermiljøet.

Tilbakemelding fra oppdragsgiver var svært positive, oppdragsgiver påpekte at det hadde blitt utviklet gode løsninger for systemet og at modellen virket svært lovende med utgangspunkt i omfanget av datasettet brukt til trening.

7.2 Videre arbeid

7.2.1 Integrering i sensorpakke

Videre arbeid for utvikling og forbedring av systemet vil være å integrere programsystemet sammen med sensor systemet. Ved full integrering kan hele systemet bli testet i reell brukssituasjon. I tillegg vil det være nyttig for å samle inn mer treningsdata fra reelle situasjoner. Det er også mulig å spesialisere en individuell modell per lokasjon med nok data.

7.2.2 Anbefaling for videre arbeid med YOLOv5 modell

Det er stort forbedringspotensial for modellen ved å utvide eller spesialisere datasettet.

Da majoriteten av viltpåkjørsler skjer om natten burde bilder av hjort fotografert om natten inkluderes i datasettet. Et annet alternativ er at modellen trenes på termiske eller infrarøde bilder i stedet for eller i tillegg til fargebilder. Dette gir bedre vesentlig høyere deteksjon av hjort når det er mørkt ettersom det er høyere kontrast. Dette vil og ha fordeler ved deteksjon på dagtid da det blir lettere å detektere hjort gjennom tett skog.

Systemet burde også trenes opp på resterende typer hjortevilt, elg, rein og rådyr. Dette vil gjøre det mulig å ta systemet i bruk i flere bruksmiljø hvor dyrene ferdes samtidig for å bidra til å redusere kollisjoner med respektive dyr. Spesielt elg hvor skadeomfanget ved påkjørsel er stor er viktig å inkludere i treningssettet.

7.2.3 Anbefaling for videre arbeid programsystemet Realtime-DWLS

Programsystemet levert av prosjektet har og stort forbedringspotensial ved videre utvikling.

Det kan utvikles et *Active Vision* system hvor programsystemet benytter høyere oppløsning fra kamera, bildet deles så opp i mindre deler som programsystemet sender til modellen for prediksjon.

Muligheter å tilrettelegge implementasjon av andre sensorer som ikke sender data i form av bilder, eksempelvis radar og bevegelsessensor. Dersom programsystemet kan ta inn data fra forskjellige typer sensorer vil det gjøre det mulig å veksle mellom ulike sensorer og varsle dersom en eller flere av sensorene detekterer hjortevilt.

Det kan implementeres en bevegelsessensor koblet opp mot programsystem og sensorsystem. Ved bruk av bevegelsessensor kan det være mulig å implementere en funksjon som sørger for at programsystemet stopper når ingen bevegelse blir detektert, dette vil føre til at modellen bruker mindre strøm. Ettersom systemet skal stå ute i felt og

potensielt ikke har fast strømtilkobling er det viktig at modellen er så energieffektiv som mulig.

Det kan videreutvikles dagens løsning over til en total skytjeneste hvor sensorer sender data til en web-applikasjon hvor modellen kjører og får respons med varsel. Dette vil føre til lavere krav til maskinvare da prosesseringen vil ikke lenger bli utført i lokalt. Dette vil gjøre det mulig å nytte tyngre modeller for detektering samtidig som det blir er enklere å vedlikeholde programsystemet da en ikke trenger dra ut til hver enkelt enhet for å oppdatere den fysiske enheten.

Det er og sannsynlig at et varslingssystemet består av flere sensorer. Der er derfor anbefalt å videreutvikle programsystemet til å ta inndata fra flere sensorer samtidig.

Loggføring og lagring av deteksjoner i en database vil gjøre det mulig å kartlegge trafikk ved de forskjellige varslingspunktene, dette kan gi innsikt om det må gjøres eventuelle tiltak for å forbedre sikkerhet og er derfor en funksjon som anbefales for videre utvikling.

8 REFERANSELISTE

Google Colaboratory (2019). Google Colaboratory. Tilgjengelig på: https://colab.research.google.com/. (Hentet 10. Mars 2022)

Hegland, S. J. (2021, May 10). Hjortebading Vill Vest. YouTube. Tilgjengelig på: https://www.youtube.com/watch?v=kjBVk6-cHcM (Hentet 16. Mai 2022)

Hegland, S. J. (2021, May 10). Hjortesisten Vill Vest. YouTube. Tilgjengelig på: https://www.youtube.com/watch?v=g0BTktf5mEk (Hentet 16. Mai 2022)

Hegland, S. J., og Hamre, L. N. (2018). Scale-dependent effects of landscape composition and configuration on deer-vehicle collisions and their relevance to mitigation and planning options. Landscape and Urban Planning, 169, 178- 184. https://doi.org/https://doi.org/10.1016/j.landurbplan.2017.09.009

Hegland, S. J. og Frøyen, M. (2021). Evaluering av forsøk med dynamiske hjorteskilt i Vestland fylke 2017-2021. hvlopen.brage.unit.no. Tilgjengelig på: https://hvlopen.brage.unit.no/hvlopen-xmlui/handle/11250/2834054 (Hentet 26. Februar 2022).

Hui, J. (2019). Object detection: speed and accuracy comparison (Faster R-CNN, R-FCN, SSD, FPN, RetinaNet and…. Medium. Tilgjengelig på: https://jonathan-hui.medium.com/object-detection-speed-and-accuracy-comparison-faster-r-cnn-r-fcn-ssd-and-yolo-5425656ae359 (Hentet 26. Februar 2022).

Jetson Nano Developer Kit. NVIDIA Developer. (2021, April 14). Tilgjengelig på: https://developer.nvidia.com/embedded/jetson-nano-developer-kit (Hentet 16. Mai 2022)

Kaarud, J. T., Nordvik, M. F., & Paulsen, H. R. (2020). Drone-based Detection of Sheep using Thermal and Visual Cameras: A Complete Approach (Master's thesis, NTNU).

Ken Schwaber. (2002) Agile software development with Scrum, Upper Saddle River, Nj Prentice Hall

makesense.ai. (u.å.). Make Sense. Tilgjengelig på: https://www.makesense.ai/. (Hentet 26. Februar 2022)

Nguyen, N.-D., Do, T., Ngo, T.D. and Le, D.-D. (2020). An Evaluation of Deep Learning Methods for Small Object Detection. Journal of Electrical and Computer Engineering. Tilgjengelig på: https://www.hindawi.com/journals/jece/2020/3189691/. (Hentet 26. Februar 2022)

NVIDIA Developer. (2013). CUDA Toolkit. Tilgjengelig på: https://developer.nvidia.com/cuda-toolkit. (Hentet 15. Mai 2022)

Olson, M. og Holmberg, I. (u.å.). Faunapassage i plan med viltvarningssystem. Trafikverket.

OpenCV Introduction. OpenCV. (u.å.). Tilgjengelig på: https://docs.opencv.org/4.x/d1/dfb/intro.html (Hentet 15. Mai 2022)

Padilla, R., Netto, S. and da Silva, E. (2020). A Survey on Performance Metrics for Object-Detection Algorithms. Conference: 2020 International Conference on Systems, Signals and Image Processing. Tilgjengelig på: https://www.researchgate.net/publication/343194514_A_Survey_on_Performance_Metrics_for_Object-Detection_Algorithms (Hentet 22. Februar 2022)

paperswithcode.com. (u.å.). Papers with Code - PASCAL VOC 2007 Benchmark (Real-Time Object Detection). Tilgjengelig på: https://paperswithcode.com/sota/real-time-object-detection-on-pascal-voc-2007 (Hentet 15. Mai 2022)

Pershona, Zhanna. Kazdorf, Sergei. (2019) Development and Research of Hand Segmentation Algorithms on the Image Based on Convolutional Neural Networks. Tilgjengelig på: https://www.researchgate.net/publication/332309907_Development_and_Research_of_Hand_Segmentation_Algorithms_on_the_Image_Based_on_Convolutional_Neural_Networks (Hentet 6. Mai 2022)

Python 3.10.4 documentation. 3.10.4 Documentation. (u.å.). Tilgjengelig på: https://docs.python.org/3/ (Hentet 20 Mai 2022)

pytorch.org. (u.å.). PyTorch. Tilgjengelig på: https://pytorch.org/hub/ultralytics_yolov5/. (Hentet 10. Mars 2022)

Qengineering. (u.å.). Qengineering/Yolov5-NCNN-raspberry-PI-4: Yolov5 for a bare raspberry pi 4. GitHub. Tilgjengelig på: https://github.com/Qengineering/YoloV5-ncnn-Raspberry-Pi-4 (Hentet 6. Mai 2022)

raspberrypi.com. (u.å.). Raspberry Pi Documentation - Compute Module Hardware. Tilgjengelig på: https://www.raspberrypi.com/documentation/computers/compute-module.html#datasheets-and-schematics (Hentet 27. Februar 2022)

Raspberry Pi Foundation (2019). Raspberry Pi — Teach, Learn, and Make with Raspberry Pi. Raspberry Pi. Tilgjengelig på: https://www.raspberrypi.org/. (Hentet 10 Mars 2022)

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., & Fei-Fei, L. (2015). Imagenet Large Scale Visual Recognition Challenge. International Journal of Computer Vision, 115(3), 211–252. https://doi.org/10.1007/s11263-015-0816-y (Hentet 13. Mars 2022)

Skara, A.B. (2019). 8 800 hjortevilt drept I Trafikken. ssb.no. Tilgjengelig på: https://www.ssb.no/jord-skog-jakt-og-fiskeri/artikler-og-publikasjoner/rekordmye-hjortevilt-drept-i-trafikken--391101 (Hentet 9. Mars 2022)

SSB. (u.å.). Flere rådyr påkjørt og drept. Tilgjengelig på: https://www.ssb.no/jord-skog-jakt-og-fiskeri/jakt/statistikk/registrert-avgang-av-hjortevilt-utenom-ordinaer-jakt/artikler/flere-radyr-pakjort-og-drep (Hentet 20 Mai 2022)

Srivastava, S., Divekar, A.V., Anilkumar, C., Naik, I., Kulkarni, V. and Pattabiraman, V. (2021). Comparative analysis of deep learning image detection algorithms. Journal of Big Data, 8(1). Tilgjengelig på: https://journalofbigdata.springeropen.com/articles/10.1186/s40537-021-00434-w#Sec26 (Hentet 10. Mars 2022)

Tale Skjølsvik and Kari Håvåg Voldsund (2016). Forretningsforståelse. Oslo Cappelen Damm Akademisk, p.344.

Tkinter - Python interface to TCL/TK. tkinter - Python interface to Tcl/Tk - Python 3.10.4 documentation. (u.å.). Tilgjengelig på: https://docs.python.org/3/library/tkinter.html (Hentet 16. Mai 2022)

Trello. (u.å.). Tilgjengelig på: https://trello.com (Hentet 2. Mars 2022)

Ultralytics (2020). ultralytics/yolov5. Tilgjengelig på: https://github.com/ultralytics/yolov5 (Hentet 17. Februar 2022)

viso.ai. (2021). Google Coral for Computer Vision Applications in 2022. Tilgjengelig på: https://viso.ai/edge-ai/google-coral/ (Hentet 26. Februar 2022)

Wallace J. Hopp and Mark L. Spearman (2002-2003) TO PULL OR NOT TO PULL: WHAT IS THE QUESTION?, Department of Industrial Engineering and Management Sciences Northwestern University

9 VEDLEGG

Vedlegg 1 Visjonsdokument

Vedlegg 2 Prosjekthåndbok

Vedlegg 3 Kravdokumentasjon

Vedlegg 4 Systemdokumentasjon

10 ORDLISTE

Ord	Forklaring
Algoritme	Er en nøyaktig beskrivelse av fremgangsmåten for en oppgave.
Avgrensningsbokser	Opptegnet boks som definerer hva som er ansett som et objekt.
CUDA	Programvare som lar grafikkprosessor ta over store deler av prosesseringen.
CPU	Prosessoren til datamaskinen.
Datasett	Samling av bilder som blir delt opp trenings- og valideringssett
Deteksjon	Gjenkjennelse av objekt.
Dyplæring	Maskinlæring-algoritme som er strukturert i nevrale nettverk.
Ekte positiv/Ekte negativ	Et utfall av hvor modellen gir en korrekt deteksjon.
Falsk positiv/Falsk negativ	Et utfall av hvor modellen gir en feil deteksjon.
Frames Per Second, FPS	Bilder per sekund.
Grafiske grensesnittet (GUI)	Det bruker ser av programsystemet.
GPU	Grafikkprosessor.
Iterasjoner	Forskjellige faser i utviklingsprosessen.
Intersection over Union (IoU)	Metode for å sjekke nøyaktighet på en detektering av objekt ved å sjekke hvor mye predikert plassering overlapper med sann plassering.
Inndata	Data som kommer utenfra til programsystemet.
Jetson Nano	Liten, kraftig datamaskin utviklet for bruk av maskin- og dyp lære applikasjoner som har CUDA maskinvare.
JSON	JavaScript Object Notation, en måte å formatere en tekststreng på.
Konfidensverdi	Modellens konfidens på en gitt deteksjon. Konfidensverdi

Ord	Forklaring
	angir hvor sikker modellen er på en deteksjon.
Ledetekst (CMD/Command Prompt)	Tekstbasert brukergrensesnitt. Brukes for å kjøre kommandoer eller utføre avanserte funksjoner.
Maskinvare	Fysiske komponenter programsystemet kjøres på.
Mean average precision (mAP)	Utrekning brukt til å måle gjennomsnittlig presisjon av en dyplærings algoritme
Millisecond, MS	Millisekund.
Modell	Resultatet av å trene en algoritme på et datasett.
MQ Telemetry Transport (MQTT)	Nettverksprotokoll for maskin-til-maskin-kommunikasjon.
OpenCV	Rammeverk for Python
Parametere	Verdier brukt til å kontrollere en modells opplæringsprosess.
PC	Personal Computer, personlig datamaskin.
Pipeline	En kjede av prosesser hvor resultatet fra prosess A brukes i prosess B, som brukes i prosess C.
Precision	Presisjon.
Python	Programmeringsspråk.
Raspberry Pi	En enkel, liten og kostnadseffektiv modulær datamaskin.
Recall	Antallet riktige deteksjoner av det totale antallet objekter.
Sanitering	Rensking av data for feil eller mangler med hensikt å øke kvaliteten.
Termisk eller infrarødt	Type kamera som spesialiserer seg på filming i mørket.
Treningsdata	Mindre del av treningssettet som blir brukt til å trene modellen.
Treningssett	En del av datasettet som bare blir brukt til trening
URL (Uniform Resource Locator)	En “adresse” til en nettside.

Ord	Forklaring
USB-akselerator	En USB som kan kobles til en pc, kan hjelpe med prosessering av bilder.
Use case	Planlagt brukstilfelle for programsystemet.
Valideringssett	Sett med bilder brukt for og teste modellen med etter trening.
Viltsluse	Tilrettelagt krysningspunkt ved vei for vilt.