Høgskulen på Vestlandet

BACHELOROPPGAVE:

BO24EH-02 APPETITTSTYRING FISK KI

Christoffer Bø

Victor Jørgensen

20. mai. 2024

Dokumentkontroll

Rapportens tittel: BO24EH-02 Appetittstyring Fisk KI	*Dato/Versjon* 20.mai.2024/1.00
	Rapportnummer: BO24EH-02
Forfatter(e): Christoffer Bø Victor Jørgensen	*Studieretning:* AUTYH_2021
	Antall sider m/vedlegg -
Høgskolens veileder: Harald Spångberg	*Gradering:* Åpen
Eventuelle Merknader: Vi tillater at oppgaven kan publiseres.	

Oppdragsgiver: Helgevold Elektro AS	*Oppdragsgivers referanse:* Ole Ronny Andersen
Oppdragsgivers kontaktperson(er) (inklusiv kontaktinformasjon): Ole Ronny Andersen – ora@helgevold.com Andrius Jasiulionis – andrius@helgevold.com	

Forord

Tidlig i studieløpet ble vi enige om å skrive bacheloroppgaven sammen. Vi brukte høsten 2023 på å finne frem til en passende oppgave, og i denne prosessen ble det holdt møter med Helgevold for idédiskusjon. Disse møtene førte oss til denne spennende oppgaven. Bedriftens ønske var å få bedre innsikt innenfor kunstig intelligens, og potensielle bruksområder.

Vi er fornøyde med valget av oppgave, da prosjektet har vært både variert, spennende og lærerik. Gjennom vårt bachelorprosjekt har vi tilegnet oss verdifulle erfaringer og kompetanse som vil være nyttige for videre arbeid innenfor automasjonsfaget.

En stor takk rettes til alle som har bidratt og gitt støtte gjennom bachelorprosjektet:

- Høgskulen på Vestlandet, for et godt studieløp.
- Intern veileder Harald Spångberg for god oppfølging og rådgivning underveis.
- Ekstern veileder Andrius Jasiulionis for rådgivning og tips til utvikling i prosessen.
- Avdelingsleder for automasjon Ole Ronny Andersen for støtte og stort engasjement for oppgaven.
- Helgevold, for muligheten til å skrive oppgave hos dem. Det har vært en stor fordel for oss å få kontorplass til å kunne arbeide med prosjektet i deres lokaler. En felles takk til alle ansatte som har spurt spørsmål rundt oppgaven, og til de som har gitt råd og hjelp underveis.

Sammendrag

Dokumentet beskriver et bachelorprosjekt som har hatt fokus på utvikling av et system for appetittstyring hos fisk ved bruk av kunstig intelligens (KI). Prosjektet tar sikte på å forbedre fôringsprosesser i fiskeoppdrett ved å redusere overfôring, noe som kan skade miljøet og føre til økte kostnader.

Hovedpunkter:

- Problemstilling: Overfôring i fiskeoppdrett og dens miljøpåvirkning.
- Kravspesifikasjon: Utvikle et system som kommuniserer med fôringssystemet for å pause fôring ved deteksjon av overfôring.
- Løsningsforslag: Bruk av kunstig intelligens og objektdeteksjon for å identifisere og telle pellets, og dermed regulere fôrmengden.
- Tekniske Komponenter: NVIDIA Jetson Orin Nano DK 8GB, Siemens S7-1512C-1 PN, og diverse programvare for KI og maskinlæring.
- Valg av Løsning: Bruk av eksisterende kamera i merd for å samle data og trene KI-modellen, som anses som kosteffektivt og lett å integrere i eksisterende systemer.

Bruk av kunstig intelligens som metode for å redusere overfôring i fiskeoppdrettsnæringen har potensiale til å både effektivisere forbruke, og gi redusert klimafotavtrykk. Oppgaven belyser videre nødvendig arbeid som må til for at metoden kan tas i bruk i praksis.

Innhold

Ordforklaringer og forkortelser

Ord / Forkortelse	Forklaring
.py	Navnet på et pythonskript slutter med .py. Eksempelvis Client.py
API	Application Programming Interface
Batch-size	Antall treningsprøver som behandles før modellen oppdateres
Bounding Box	Boks med x-y koordinat som blir markert rundt et objekt I bilde
Capture-Tool	Annoteringsverktøy inkludert med Jetson-Inference
Epoch	En runde med trening og validering gjennom algoritmen
Flashe	Oppdatere eller installere programvare
Good fit	Modellen klarer å generalisere godt fra treningsdata
GPU	Graphical Processing Unit
GUI	Graphical User Interface
IDE	Integrated Development Environment
Image	Operativsystem eller programvare formatert til å bli flashet på en enhets minne
IP-kamera	Internet Protocol Kamera
KI	Kunstig Intelligens
Launcher	Program eller applikasjon som starter opp andre programmer
Localization Error	Når en modell korrekt identifiserer et objekt, men bommer på lokaliseringen
Merd	Innhegning av oppdrettsfisk
Mikrokontroller	Integrert krets som inneholder prosessor, funksjonsblokker, minne og inn-ut-enheter.
Modbus	Klient/Server datakommunikasjonsprotokoll
Model fit	Hvor godt en modell samsvarer med de faktiske dataene
MQTT	Nettverksprotokoll
OBS	Open Broadcaster Software
Overfitting	Når en modell lærer seg detaljer og støy til det punkt hvor det påvirker negativt
Pellet	Små sylindre som er laget ved å presse sammen materie(fôr)
PLS	Programmerbar Logisk Styring
Pythonskript	Kode skrevet i Python
QSPI	Quad Serial Peripheral Interface
R-CNN	Region-based Convolutional Neural Network
Recall	Hvor stor andel av faktiske positive ble identifisert korrekt
SD-kort	Secure Digital, minnekort
SSD	Single Shot Multibox Detectors
TCP	Transmission Control Protocol
Underfitting	Når en modell er for enkel til å lære strukturen av dataene
Workers	Antall parallelle prosesser eller tråder som brukes for å laste data
YOLO	You Only Look Once

Figurliste

1 Innledning

1.1 Oppdragsgiver

Helgevold er en av de ledende automasjon- og elektroentreprenørene på Haugalandet. Gjennom sterke fagmiljøer innen elektro, automasjon og telekommunikasjon leveres gjennomtenkte løsninger av elektriske anlegg. Helgevold Elektro AS ble etablert i 1987. Gjennom vekst og oppkjøp har selskapet nå vokst til en organisasjon på over 300 ansatte. Det er en lokalt forankret bedrift med 11 lokasjoner spredt utover Haugalandet og omegn. Automasjonsavdelingen til Helgevold ble etablert for over 30 år siden og er i dag et team med over 30 ansatte. Der går det i store oppdrag innenfor blant annet bygg automasjon, industri og fiskeoppdrett. Her arbeides det med prosjektering, tegning, programmering og idriftsettelse [1].

1.2 Problemstilling

Problemstillingen omfatter fiskeoppdrettsnæringen og overfôring i forbindelse med fôring av laks i sjøoppdrett. Overfôring er noe som kan skade det ytre miljø utenfor fiskeoppdrettet. Ved overfôring blir det en stor samling av fiskefôr og slam på havbunnen, som legger seg som et tykt lag over den opprinnelige havbunnen. Dette kan ødelegge økosystemet som lever der, som igjen kan føre til en ubalanse på den norske kyst og spesielt i de norske fjordene [2]. Dersom det blir laget et system for dette vil det spare miljøet for utslipp, og oppdrettsnæringen vil kunne kutte kostnader i forbindelse med fiskefôr. Fôr utgjør opptil 83 prosent av klimagassutslipp for norsk laks [3].

1.3 Kravspesifikasjon

Systemet skal kunne kommunisere med fôringssystemet for å stanse og pause fôringen ved oppdagelse av overfôring av fisk. Oppdragsgiver kom med noen veiledende kravspesifikasjoner:

- Systemet skal kunne styre fôring på fiskeoppdrett eller simulere det ved hjelp av mikrokontroller og Programmerbar Logisk Styring (PLS).
- Styring av fiskefôr skal være basert på deteksjon av pellets i video ved hjelp av kunstig intelligens/objekt deteksjon i video/bilde.
- Systemet skal ha kommunikasjon fra mikrokontroller til PLS eller PC.
- Trening av signaler og styring mellom PLS og mikrokontroller

1.4 Hovedidé for løsningsforslag

Det skal brukes en mikrokontroller med KI og objektdetektering. Mikrokontrolleren skal brukes til å identifisere og telle antall pellets som går forbi beltet med fisk. Her må det brukes kamera for deteksjon. Eksisterende kamera i merden kan brukes, eller så kan det løses med eksternt kamera. Dette skal brukes for å kunne regulere mengden fiskefôr som blir sendt ut i merden. Det som skal til for å kunne gjøre dette mulig er å programmere mikrokontrolleren, og ved hjelp av maskinlæring skal den kunne «kjenne igjen pellets». Løsningsforslaget innebærer også at det blir opprettet kommunikasjon som sender signal fra mikrokontroller til en ekstern PLS, eller datamaskin med en programvare. Begge løsningene må ha en mulighet for å behandle signalene fra mikrokontrolleren. Her bør det være en alarm eller varsling som blir gitt ved overfôring og fôring pauses en god stund før den starter opp igjen.

2 Teori

Kapitelet vil gi et teoretisk innblikk i kunstig intelligens, maskinlæring, objektdetektering og annen teori knyttet til dette.

2.1 KI, Maskinlæring og Objektdetektering

Kunstig intelligens er teknologi som justerer sin egen aktivitet, og opptrer dermed intelligent. En datamaskin med kunstig intelligens kan løse oppgaver uten direkte menneskelige instruksjoner. For eksempel, intelligente søkemotorer bruker tidligere søk og brukeratferd til å foreslå relevante resultater, en prosess kjent som maskinlæring. Det forskes mye på KI i felt som språkteknologi, talegjenkjenning, bildegjenkjenning, brukerinteraksjon og prosesskontroll. Vanligvis refererer man til dype nevrale nettverk når man snakker om KI [4].

Objektdeteksjon er en teknikk innen datavisjon for å lokalisere forekomster av objekter i bilder eller videoer. Algoritmer for objektgjenkjenning benytter vanligvis maskinlæring eller «deep learning» for å oppnå meningsfulle resultater. Når mennesker ser på bilder eller videoer, kan objekter gjenkjennes og lokaliseres. Målet med objektdetektering er å etterligne denne intelligensen ved bruk av en datamaskin [5].

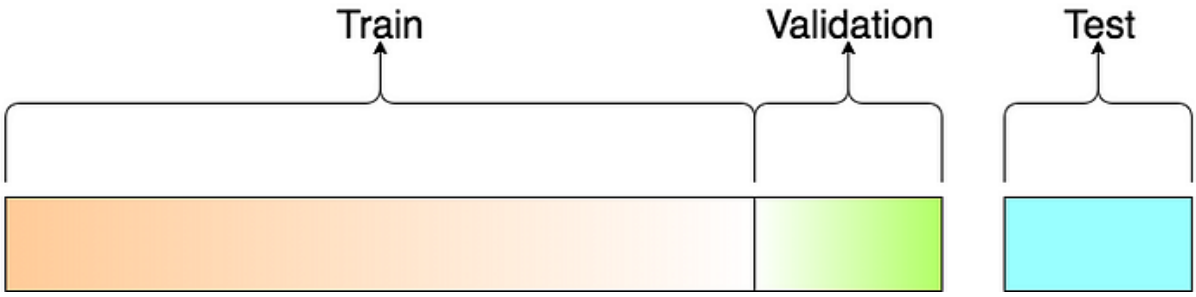
2.2 Bildegjenkjenningsmodell

En bildegjenkjenningsmodell er en type maskinlæringsmodell som er designet for å identifisere og klassifisere objekter, personer, handlinger og mer i digitale bilder. Disse modellene bruker algoritmer for å analysere bildeinnhold, og algoritmene kan skille mellom ulike visuelle elementer basert på lærte egenskaper fra store datasett med merkede bilder. De blir brukt til blant annet sikkerhetssystemer, autonom kjøring, medisinsk bildediagnostikk, med mer. Forskjellige bildegjenkjenningsarkitekturer som er utbredt er YOLO, R-CNN og SSD [6]. For å lese mer om SSD se kapittel 4.5.2.1.

2.3 Treningsdata, valideringsdata og testdata

Det brukes en annoteringsprogramvare for å lage treningsdata der brukeren må plassere posisjonen til objektene, det samme gjelder for valideringsdata og testdata.

Treningsdata er et sett med bilder som har tilhørende posisjonsdata til en objektklasse. Objektklasse er en klasse som inneholder posisjonsdata til et navngitt objekt. Posisjonene som er i de ulike objektklassene blir brukt til trening av bildedeteksjonsmodellen [7].



Figur 1- Visualisering av størrelse av datasettene. Hentet fra https://towardsdatascience.com/train-validation-and-test-sets-72cb40cba9e7

Valideringsdata er tilsvarende som treningsdata, forskjellen er at valideringsdataen er vesentlig mindre enn treningsdataen, se Figur 1. Valideringsdataen blir brukt i trening av modellen for å evaluere «model fit» på treningsdatasettet. «Model fit» er hvor godt en modell samsvarer med de faktiske dataene. Valideringen er også til god hjelp for evaluering av resultater av treningen, og nyttig under utvikling av modellen [8].

Testdata er bilder som blir brukt til en sluttevaluering av modellen fra treningsdataen. Dette er data som skal inneholde bilder som er helt separat fra trening og validering med større variasjon i bildemotiver. Det er for å gi modellen mer utfordring for en skikkelig sluttevaluering av modellen som er trent [8].

2.4 Evaluering av trening

Ved trening av en bildedeteksjonsmodell blir epochs ofte omtalt. Epochs er “trenings epoker” som vil si hvor mange ganger datasettene blir kjørt igjennom trening. Når trening av «epoch 0» er ferdig, så trenes denne modellen en gang til i «epoch 1». Sånn fortsetter den til den har vært gjennom alle «epochs» i treningen. Hver «epoch» er komplette modeller, og den «epochen» med minst tap i trening og validering blir valgt ut som endelig bildedeteksjonsmodell.

Tapet i trening og validering kan bli brukt til å lage læringskurver. Ved trening av en bildedeteksjonsmodell, er ønsket at tapet på trening og validering er nærmest mulig 0, og mest mulig lik i verdi. Da klarer bildedeteksjonsmodellen lettere å generalisere ny data under trening. Videre er det mulig å analysere hvordan grafene oppfører seg dersom det er en «overfitting», «underfitting» eller «good fit» bildedeteksjonsmodell.

Overfitting er når bildedeteksjonsmodellen har lært treningsdatasettet for godt, noe som resulterer i at modellen ikke lærer på ny data i treningen. Fordelingen i datasettene må omstruktureres for å oppnå en bedre trening av bildedeteksjonsmodellen. På grafene kan det identifiseres en overfit når «train» grafen synker, men valideringsgrafen er relativt stabil.

Underfitting er når bildedeteksjonsmodellen ikke klarer å lære fra treningssettet, eller at treningsintervallet med antall epochs er for kort. Det kan løses med et lengre trenings intervall (flere epochs), eller undersøke annoteringen i treningsdatasettet nærmere. Læringskurven på underfitting, kan vise en flat graf eller en graf med mye støy og kan identifiseres bare ved å se på treningsgrafen.

Good fit er når trening og validering synker til ett lavt tap der mellomrommet mellom tap i trening og validering er lav. Grafisk vises dette med at verdiene synker, flater ut, og nærmer seg samme verdi [9]. Målsettingen med plot av tap er å oppnå good fit, fordi da har man fått mest mulig effektiv trening av datasettene.

2.5 Presisjon og Recall på bildedeteksjonsmodell

Presisjon og «Recall» er teoretiske mål på modellen sin ytelse. Presisjon beskriver andelen relevante resultater i modellens forutsigelser. Det er en måling av hvor mange av de identifiserte positive tilfellene som faktisk var positive. Recall måler andelen av de faktiske positive tilfellene som ble korrekt identifisert av modellen. Det viser hvor godt modellen kan finne alle relevante tilfeller innenfor et datasett. Disse to målingene hjelper til med å evaluere effektiviteten av bildedeteksjonsmodeller [10].

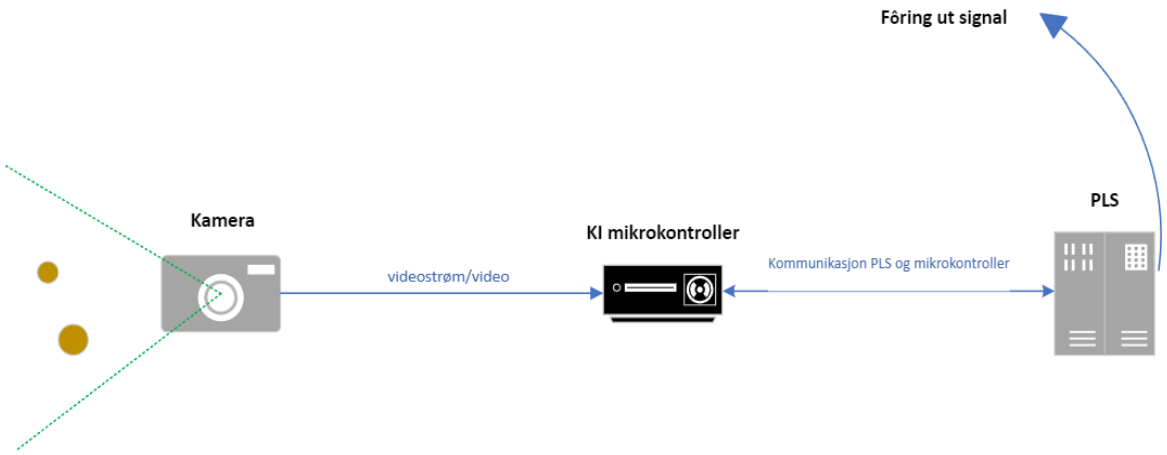
3 Utvidet oppgavebeskrivelse

Dette kapitelet tar for seg det eksisterende systemet, og ser på mulige komponenter som kan inngå i en løsning for oppgaven. Kapitelet utdyper også hvilke avgrensinger og antakelser som er gjort i løsningsforslagene. Formålet med dette prosjektet, er at det skal bli laget ett system som hindrer fôrspill og senke utslippene fra oppdrettene til det ytre miljø.

3.1 Eksisterende anlegg og løsning

Fôringen fungerer i dag ved at en operatør sitter kontinuerlig og følger med under fôring for å unngå fôrspill[11]. Det settes også en fast vekt på fôring som for eksempel 10 000 kg fiskefôr i merd nummer 4 i dag. Mengden fiskefôr er bestemt etter en gitt fôringsplan, og dette baseres på beregnet biomasse med fisk. Realiteten er at fisken ikke alltid spiser like godt, og dette kan det være forskjellige årsaker til, som for eksempel sykdom og temperatur. [12]

3.2 Utforming av mulige løsninger

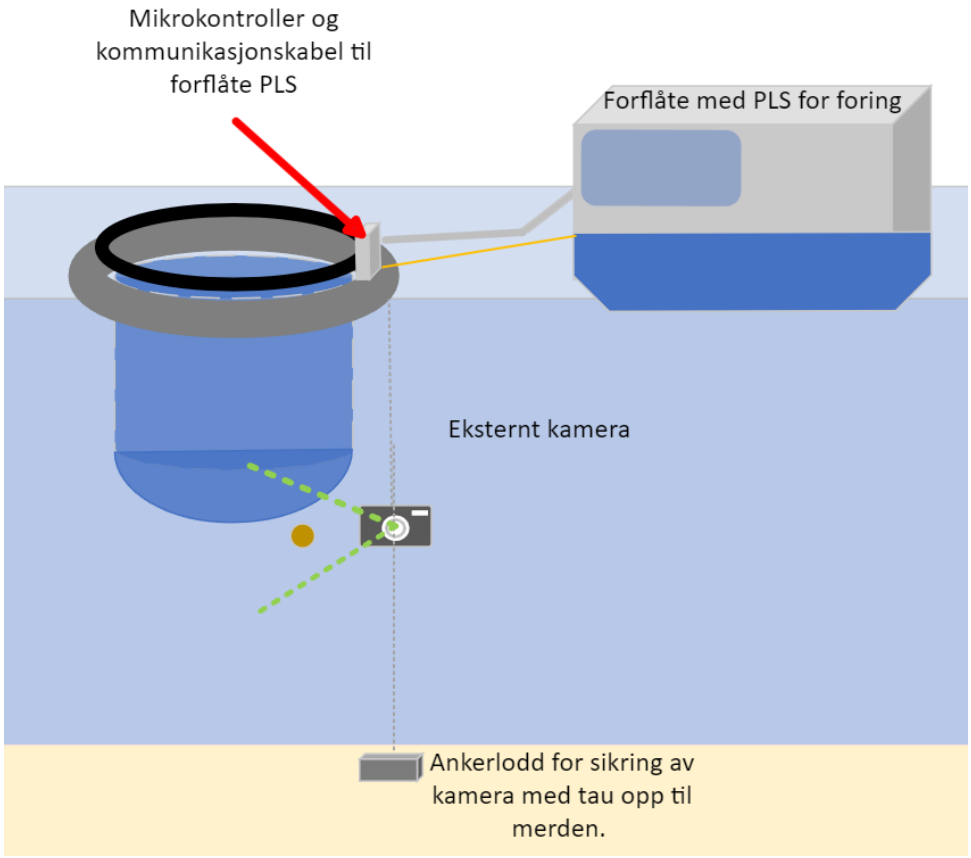


Figur 2 – Teknisk oversiktsdiagram

Figur 2 viser hvordan koblingen mellom de ulike komponentene som inngår i løsningsforslagene. Felles for alle løsningsalternativene er at det skal brukes Kamera/Video, KI mikrokontroller, bildetrening, objektdetektering, PLS, samt lage programkode til disse.

Videre er planen at fôring skal reguleres etter hvor store mengder fiskefôr som går forbi fisken uten at det blir spist. Dette er løsninger som kan spare miljøet for utslipp, og oppdrettsnæringen vil kunne kutte kostnader i forbindelse med fiskefôr.

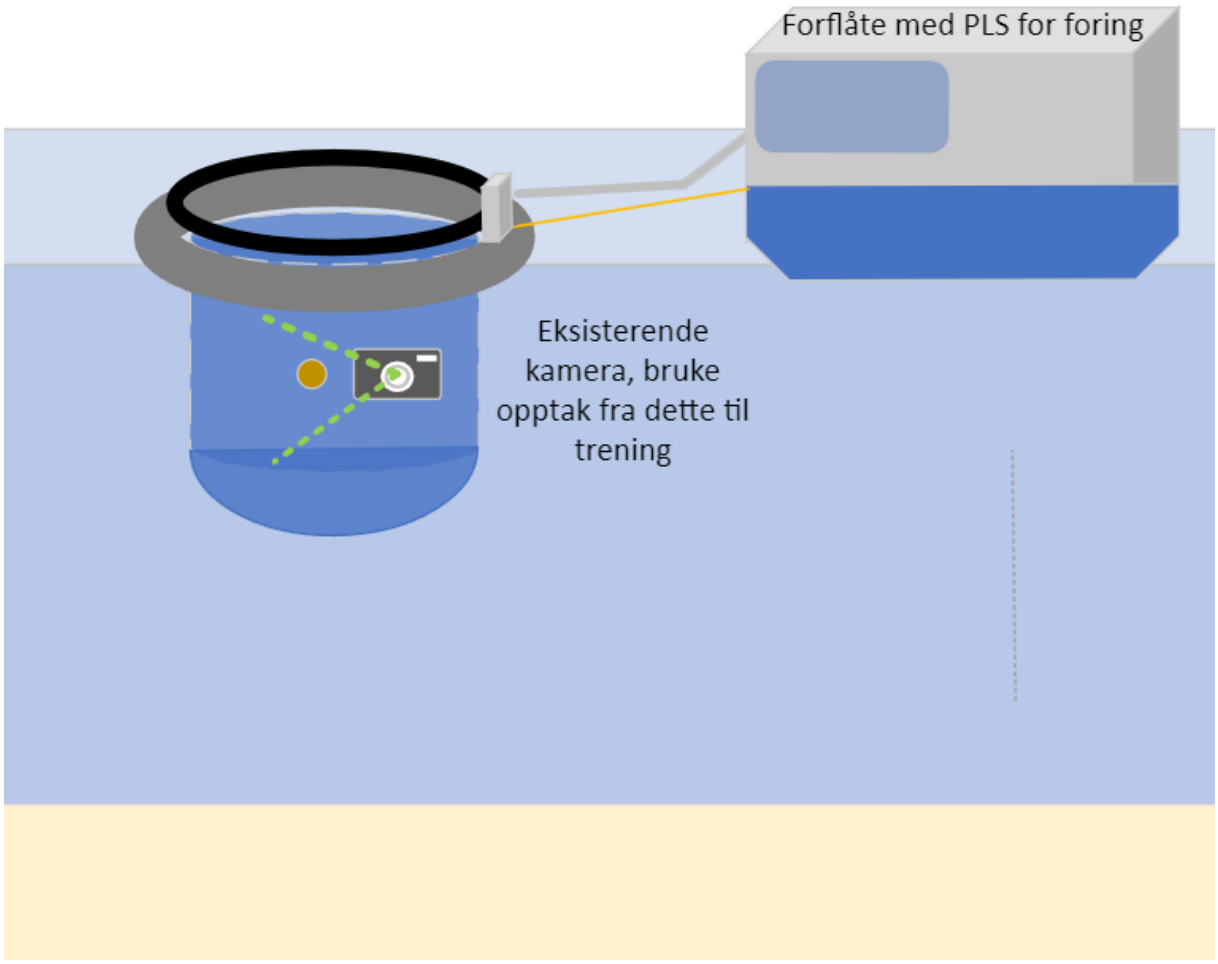
3.2.1 Løsningsalternativ 1: Bruke eksternt kamera



Figur 3 - Løsningsforslag 1

Figur 3 viser hvordan det kan løses med et eksternt kamera på utsiden av merden som ser innover mot noten. Da kan mikrokontrolleren detektere pellets som faller gjennom noten. Dette er noe som må testes, bli brukt, og tatt opptak av når fisken blir fôret. Deretter må data benyttes til å trene opp mikrokontrolleren. Løsningen krever mer design for innfestning av kamera, valg av kamera osv.

3.2.2 Løsningsalternativ 2: Bruke eksisterende kamera i merd



Figur 4 - Løsningsforslag 2

Figur 4 viser en illustrasjon over løsningsforslag 2, der løsningen er å bruke allerede eksisterende kamera i merd. Dette er som oftest et kamera som blir brukt til å observere laks under fôring. Løsningsforslaget innebærer å bruke opptak fra disse kameraene til treningen av bildedeteksjonsmodellen. Videre må det lages programkode og opprette kommunikasjonsprotokoll mellom PLS og KI mikrokontroller. I tillegg programmere PLS-en for håndtering av mottatt signal. Ved detektert overfôring, skal mikrokontrolleren sende signal til PLS-en.

3.2.3 Løsningsalternativ 3: Lage eget akvarium simulere prosessen.

Dette løsningsforslaget innebærer å bygge et eget lite testanlegg, hvor fôringen skal simuleres. Et akvarium vil bli montert med et tilhørende kamera, hvor det skal samles opptak av pellets i akvariet. Her vil lysfaktor og viskositet i vannet spille en rolle. Det er ikke sikkert denne løsningen hadde detektert pellets i en ekte laksemerd. Da det må lages så realistiske lysforhold som mulig oppi tanken. Det må også bygges et fôringssystem som styres av en PLS.

3.2.4 Vurderinger i forhold til verktøy og HW/SW komponenter

KI mikrokontroller: Skal klare å håndtere direkte videostrømming. Den må ha god programvare for objektdetektering, eller være kompatibel med programvare for objektdetektering. Dette er den viktigste komponenten i prosjektet fordi den skal detektere pellets i laksemerd, og sende signal dersom for mye pellets blir detektert. Den skal også ha mulighet for tilkobling av Ethernet. I løsningene inngår en NVIDIA Jetson Orin Nano DK 8GB, og ved programmering av mikrokontrolleren brukes Python.

PLS: Dette er styringsenheten til lakseføret. Det er denne som skal stanse fôring ved signal fra mikrokontrolleren om overfôring. I oppgaven er det ikke tilgang til en PLS med et reelt fôringssystem, men det kan tilpasses slik at det er enkelt å få gjennomført appetittstyring til et allerede eksisterende fôringssystem. For at det skal etableres kommunikasjon fra mikrokontroller til PLS, bør PLS'en ha Modbus, TCP/IP, API, MQTT, eller lignende protokoll. I løsningen inngår en Siemens-modell, forutsatt at enheten støtter nødvendig protokoll for kommunikasjon med mikrokontrolleren.

Kommunikasjon mellom mikrokontroller og PLS: Dette er en viktig del av oppgaven fordi formålet er å oppnå styring. Uten kommunikasjon mellom enhetene er det ikke mulig med styring. Det er flere kommunikasjonsalternativer å gå for. Eksempelvis «TCP/IP» som er en standard kommunikasjonsmåte for PLS, og som blir brukt i løsningen.

Trening av bilde/videosignal: Det som anses som viktig her at det blir nok datasett til å trene opp til god deteksjon av pellets.

3.3 Valg av løsning

Løsningen som ble valgt er å bruke eksisterende kamera i merd, som er nærmere beskrevet i kapittel 3.2.2. Grunnen bak valget er at denne løsningen er mer kosteffektiv, og kan brukes som en utvidelse til et allerede eksisterende system. Det finnes en del videodata fra tidligere, som kan brukes til å trene opp mikrokontrolleren. Ved å bruke eksisterende opptak vil det kunne brukes mer tid på utvikling, og ikke på å bygge testanlegg eller samle inn video fra eksternt kamera. Valget av denne løsningen gjør også at oppgaven kan gjøres mer universal, slik at oppdrettslokasjoner kan benytte seg av den uten behov for mye datainnsamling og tilpasning til ett spesifikt anlegg.

4 Tekniske komponenter og programvare i prosjektet

Dette kapitelet tar for seg tekniske komponenter og programvare som blir benyttet i prosjektet. I kapitelet beskrives det kort om hver komponent og programvare, og i noen av de blir det nevnt hvor eller hvordan de blir brukt i prosjektet. Dette kapitelet er for å gi en bedre oversikt i prosjektet.

4.1 NVIDIA Jetson Orin Nano DK 8GB

Figur 5. NVIDIA Jetson Orin Nano DK 8GB. Hentet fra https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-orin/

Jetson Orin Nano er en mikrokontroller fra NVIDIA se Figur 5, som er tiltenkt inngangsnivå for KI-styrt robotisering, smart-droner og intelligente kamera. Mikrokontrolleren er basert på et eget operativsystem som heter JetPack, som er en mer egnet versjon av Linux for KI [13]. Denne mikrokontrolleren er derfor et godt valg i forbindelse med oppgaven.

4.1.1 JetPack SDK 5.1.1

JetPack er operativsystemet til NVIDIA Jetson Orin Nano. Den inkluderer Jetson Linux 35.3.1 BSP med Linux Kernel 5.10 og Ubuntu 20.04 root file system. Noe som også inkluderes i JetPack er alle programvarer for KI-behandling av ulike varianter [14].

4.1.1.1 TensorRT

TensorRT er en optimaliseringsmotor utviklet av NVIDIA for dyp læring. Den er designet for å forbedre ytelsen til KI-modeller ved å optimalisere dem for Jetson-plattformen og dra nytte av spesifikke maskinvarefunksjoner [15].

4.1.1.2 Deep Learning Frameworks

NVIDIA støtter en rekke dype læringsrammeverk som TensorFlow, PyTorch og ONNX på Jetson-plattformen. Dette gjør det enkelt for utviklere å trene sine KI-modeller [16].

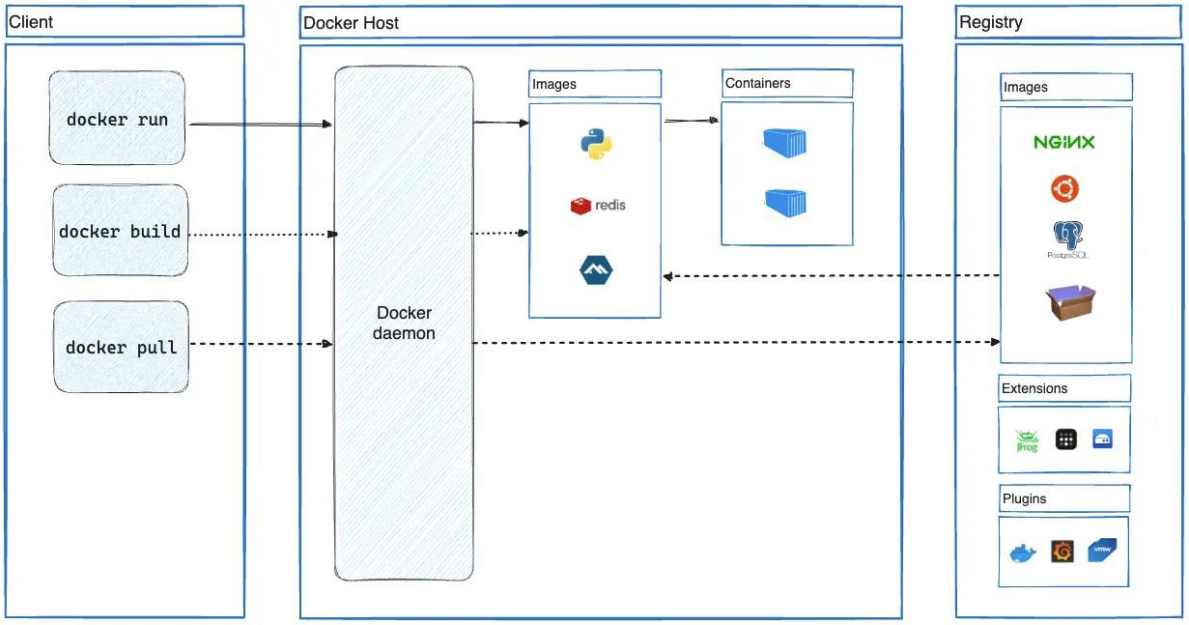
4.2 NVIDIA SDK Manager

NVIDIA Software Development Kit (SDK) Manager er et verktøy brukt til å laste ned programvare og utvidelser til Jetson-komponenter. SDK Manager installeres på en PC med Linux som operativsystem, og kommuniserer med en Jetson-komponent før den har fått programvare selv. Det er den mest anbefalte metoden å få installert operativsystem og programvare til Jetson-komponenter [17].

4.3 Jetson-Inference

Jetson-Inference er et prosjekt med åpen kildekode, utviklet av NVIDIA. Dette er en plattform som gir deg muligheter til å bygge, trene og distribuere nevrale nettverk på NVIDIA Jetson-enheter. Jetson-Inference har et sett med verktøy som gjør det mulig å utføre bildegjenkjenning, objektdeteksjon, segmentering og andre datamaskinvisjonsoppgaver i sanntid på Jetson-enheter. Den utnytter den kraftige grafikkprosessoren på Jetson-enhetene når den kjører komplekse dype nevrale nettverk [18].

4.3.1 Docker



Figur 6 - Oppbygging docker. Hentet fra https://docs.docker.com/get-started/images/docker-architecture.webp

En docker er en standardisert enhet med programvare som pakker koden, og alt tilhørende inn i en container, se Figur 6. Dette kan bli brukt for å lage programvare som kan bli kjørt som et separat system fra operativsystemet på pc-en [19]. Docker blir brukt i oppgaven ettersom Jetson-Inference er bygd i en docker-container av NVIDIA utvikleren Dustin Franklin (dusty-nv). Denne docker containeren kan lastes ned fra GitHub-en til Dusty-nv [18].

4.4 Siemens S7-1512C-1 PN

Figur 7 - Siemens strømforsyning. Hentet fra: https://mall.industry.siemens.com/mall/collaterals/files/158/jpg/P_ST70_XX_06526i.jpg

Figur 8 - Siemens S7-1512C-1 -PN. Hentet fra: https://mall.industry.siemens.com/mall/collaterals/files/158/jpg/P_ST80_XX_04005i.jpg

Figur 8 viser en S7-1512 PLS fra Siemens sin S7 serie av PLS-er. PLS-en har en analog modul og to digitale I/O moduler i tillegg til to RJ-45 kontakter for Ethernet [20]. Portene blir brukt til kommunikasjon mellom komponentene i oppgaven. Ethernet porten blir også brukt til kommunikasjon til programmeringsprogramvaren til Siemens. Det blir også brukt en strømforsyning til PLS-en fra Siemens, se Figur 7 [21].

4.4.1 TIA Portal V16

TIA Portal er en omfattende programvare for programmering og simulering av Siemens PLS'er. Her kan det programmeres i strukturert tekst eller ved hjelp av ladder [22].

4.5 Python

Python er kjent for sin klare og lesbare syntaks, som gjør det enkelt å lære. Språket brukes bredt innenfor mange forskjellige områder innenfor programmering og utvikling. Ved bruk av biblioteker og utvidelser kan man løse de fleste programmeringsutfordringer, og få god hjelp fra andre utviklere på nett [23].

4.5.1 Pythonbiblioteker

Biblioteker i Python er samlinger av moduler og funksjoner som er utviklet for å løse spesifikke oppgaver eller tilby funksjonaliteter innenfor ulike områder av programvareutvikling. Disse bibliotekene leveres vanligvis i form av installerbare pakker som kan importeres og brukes direkte i Python-programmer. Når et bibliotek er installert, bruker utviklere pakkens funksjoner ved å importere det i sine Python-skript [24].

4.5.1.1 Snap7

Snap7 er et flerplattform-bibliotek som gir muligheten til å kommunisere med Siemens S7 PLS-er via Ethernet. Biblioteket er skrevet i C, men det er tilgjengelig for flere programmeringsspråk inkludert Python, C++, C# og andre. Snap7 tillater utviklere å bygge applikasjoner som kan lese og skrive data til og fra PLS-er, samt utføre andre oppgaver som å overvåke tilstanden til PLS-er og styre produksjonsprosesser. Det er et verdifullt verktøy som knyter sammen PLS-er til applikasjoner på tradisjonelle datamaskiner og mikrokontrollere [25].

4.5.1.2 Tkinter

Tkinter er et standard GUI (Graphical User Interface) bibliotek for Python. Det gir et sett med verktøy for å utvikle brukergrensesnitt for applikasjoner på en enkel og effektiv måte. Tkinter er en del av Python sitt standardbibliotek, noe som betyr at det er tilgjengelig som standardinstallasjon med Python. Tkinter er basert på Tk GUI-toolkit, som er skrevet i TCL (Tool Command Language). Dette gjør det mulig å bygge grafiske applikasjoner ved å kombinere Python-kode med Tkinter-funksjoner som knapper, etiketter, tekst og mer [26].

4.5.1.3 DetectNet

DetectNet et pythonbibliotek utviklet av NVIDIA, og er spesielt laget for å detektere objekter i videostrømmer. Det bruker KI for å identifisere og plassere objekter i sanntid [27]. Dette biblioteket brukes i oppgaven til å laste inn SSD-bildedeteksjonsmodellen, som følger med i Jetson-Inference.

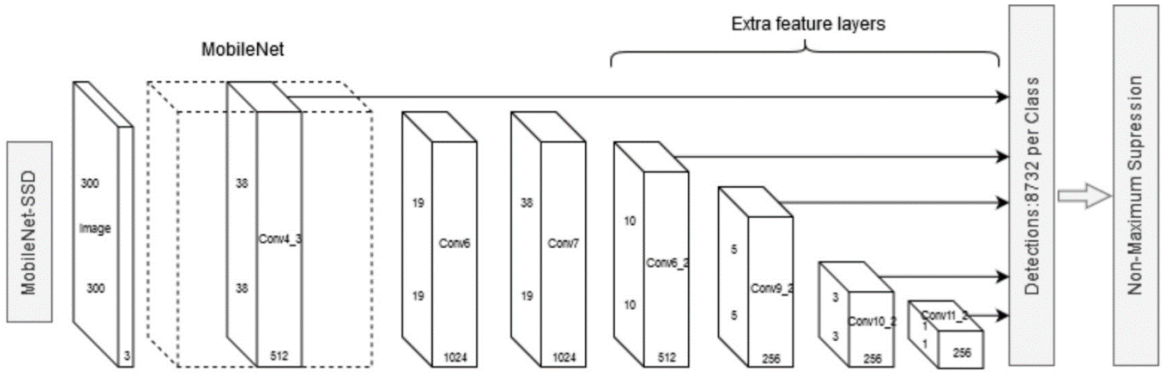
4.5.1.4 Socket

Socket-biblioteket gjør det mulig å lage en TCP-server slik at det kan bli brukt TCP/IP forbindelse mellom ulike programmer, og koble til serveren med en eller flere Client program [28].

4.5.2 Pythonbiblioteker som fulgte med Jetson-Inference

Dette er pythonbibliotekene som ikke blir brukt i prosjektets programkoder, men som er viktige bibliotek som blir brukt for treningen inne på dockeren Jetson-Inference. Jetson-Inference bruker og inneholder mange flere bibliotek, men her er de viktigste for trening av bildedeteksjonsmodeller [18].

4.5.2.1 SSD



Figur 9 - Oversikt over SSD modell

Figur 9 viser en SSD «Single-shot multibox detector» som er et nettverk for sanntids objektdeteksjon [29]. Denne modellen blir brukt med PyTorch for trening av bildedeteksjonsmodell. SSD blir brukt i denne oppgaven til å lage modeller ved hjelp av Jetson-Inference scriptet train_ssd.py [30].

4.5.2.2 PyTorch

PyTorch er et pythonbibliotek utviklet av Meta AI og det er et åpent bibliotek for maskinlæring og datavitenskap [31]. Det er dette biblioteket som blir brukt i scriptet train_ssd.py i Jetson-Inference-dockeren for trening av datasett ved hjelp av SSD.

4.5.2.3 ONNX

ONNX «Open Neural Network Exchange» er et bibliotek som støtter mange rammeverk, verktøy og maskinvarer og er optimalisert for KI modeller [32]. For å eksportere en PyTorch bildedeteksjonsmodell til ONNX-fil brukes skriptet onnx_export.py [33].

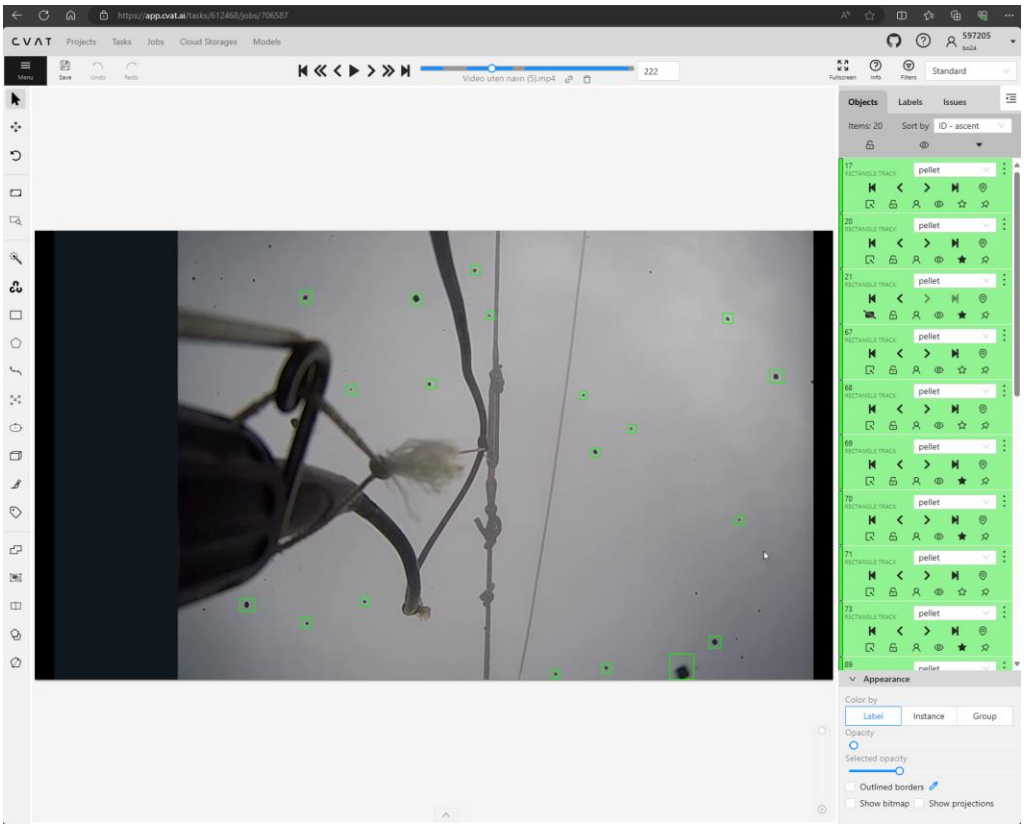
4.6 Visual Studio Code

Visual Studio Code er en tekseditor utviklet av Microsoft [34]. Den er designet for å være et enklere verktøy for programmering med støtte for de fleste språk. Med sømløs integrasjon av GitHub er det også praktisk å bruke VS Code for å programmere på forskjellige maskiner samtidig på samme prosjekt.

4.7 GitHub

GitHub er en nettbasert plattform som brukes av utviklere for deling av kode. Utviklere kan arbeide parallelt og gjennomgå kodeendringer effektivt. GitHub fungerer også som et sted hvor utviklere kan enkelt finne, utforske og laste ned andres programvareprosjekter. Denne funksjonen gjør det mulig for utviklere å dra nytte av eksisterende kodebase for å bruke til sine egne prosjekter [35].

4.8 CVAT.ai

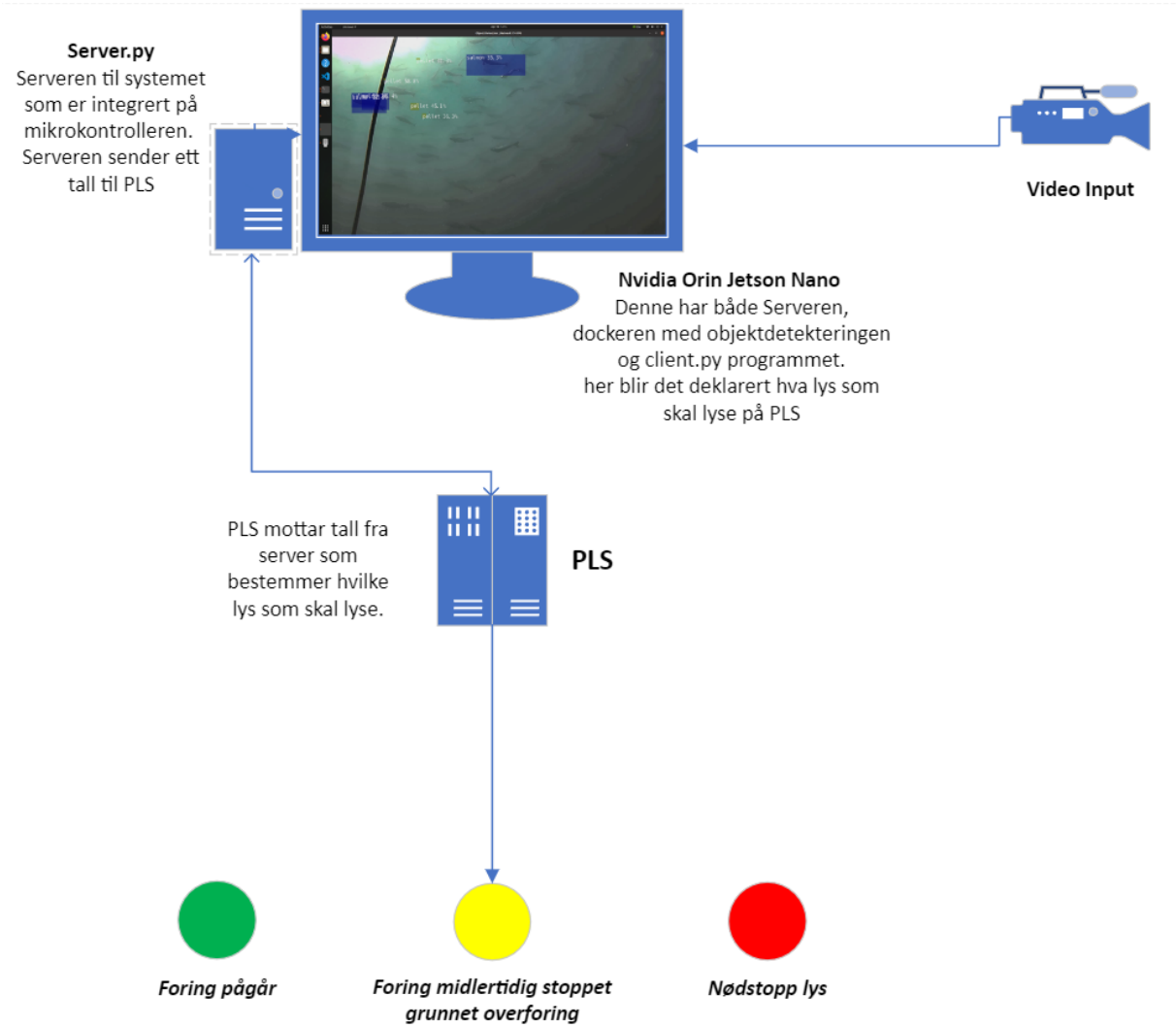


Figur 10 – CVAT skjermbilde hentet fra: https://app.cvat.ai

CVAT (Computer Vision Annotation Tool) er en åpen kildekode plattform utviklet for annotering av bilder og videoer. Den gir et brukervennlig grensesnitt for manuell annotering av objekter i bilder og videoer, og gjør det mulig for flere personer å jobbe på samme prosjekt [36]. CVAT gjør det mulig å opprette datasett og eksportere til ønsket format for bruk på Jetson-Inference. Figur 10 viser skjermbilde fra annotering av bilder i CVAT.

5 Hoveddel

I dette kapittelet skal arbeidet som er blitt utført i prosjektet beskrives. Her blir komponenter og programvaren som blir beskrevet i kapittel 4 brukt for å lage en løsning til problemstillingen til dette prosjektet.



Figur 11 - Systemoversikt

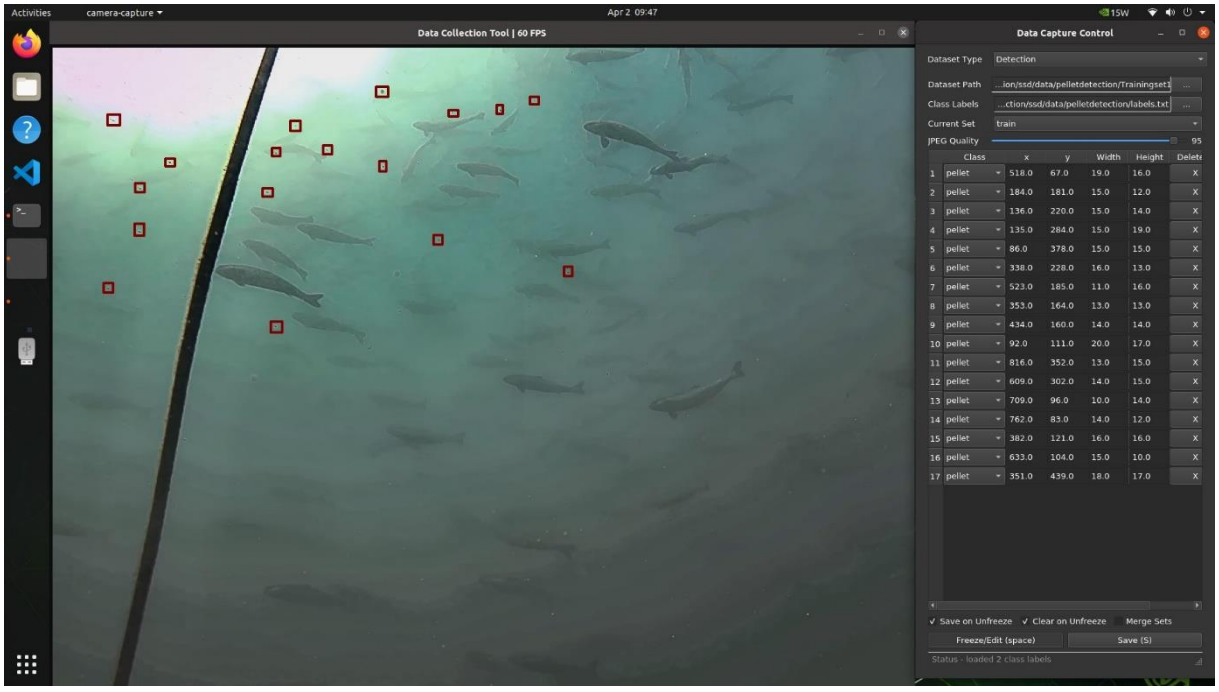
Figur 11 viser systemoversikten og gir en sammenhengende oversikt over hvordan signalene fra deteksjon sendes til slutt til PLS som gir en lysindikasjon.

Startet med å sette opp systemet av NVIDIA Jetson Orin Nano DK 8GB mikrokontroller med operativsystemet JetPack SDK 5.1.1. Operativsystemet ble installert ved hjelp av NVIDIA SDK Manager. Det var også noen utfordringer knyttet til installasjon av operativsystemet for ett nærmere innblikk, se «Utfordringer under oppsett av system».

Neste steg i arbeidet var installasjon av Docker containeren Jetson-Inference, som ble lastet ned fra GitHub via terminal på mikrokontrolleren. Se <Vedlegg 1> for oppsett av Jetson-Inference.

Da Jetson-Inference var installert startet arbeidet med å sette opp kommunikasjon mellom KI-mikrokontroller og PLS se «Beskrivelse av kommunikasjonsoppsettet». Det ble det laget to Python program og et PLS program for å opprette kommunikasjon, se «Kode». Det var noen utfordringer knyttet til IP-adresse til Docker – Jetson-Inference. For å se hva disse var, se «Utfordringer under oppsett av system».

Det ble bygget en automasjonstavle som prototype for enkel testing av system, se «Automasjonstavle».



Figur 12 - Camera-Capture Tool

Når kommunikasjonen fungerte startet annoteringen av datasett. Prøvde først «Capture-Tool» fra Jetson-Inference se Figur 12, men det fungerte ikke særlig bra. Alternativ programvare ble CVAT.ai for trening, validering og testdatasett, se Figur 10. Hele denne utfordringen står grundigere beskrevet i kapittel 8.1.

Bilder ble annotert i CVAT.ai. Dette vil si at det ble satt «bounding boxes» rundt pellets. Dette definerer objektets størrelse og posisjon i bilde. Når dette er gjort hentes ut datasettet

fra CVAT det består av alle annoteringer. Det vil si bilder med XML filer vedlagt til bilder som beskriver XY posisjonen til «bounding boksene», i tillegg til størrelsen til boksene. Formatet til datasettene heter Pascal VOC. I Figur 13 kan du se eksempel på ett objekt i formatet Pascal VOC.

```
<object>
  <name>pellet</name>
  <truncated>0</truncated>
  <occluded>0</occluded>
  <difficult>0</difficult>
  <bndbox>
    <xmin>1064.37</xmin>
    <ymin>883.09</ymin>
    <xmax>1096.69</xmax>
    <ymax>915.46</ymax>
```

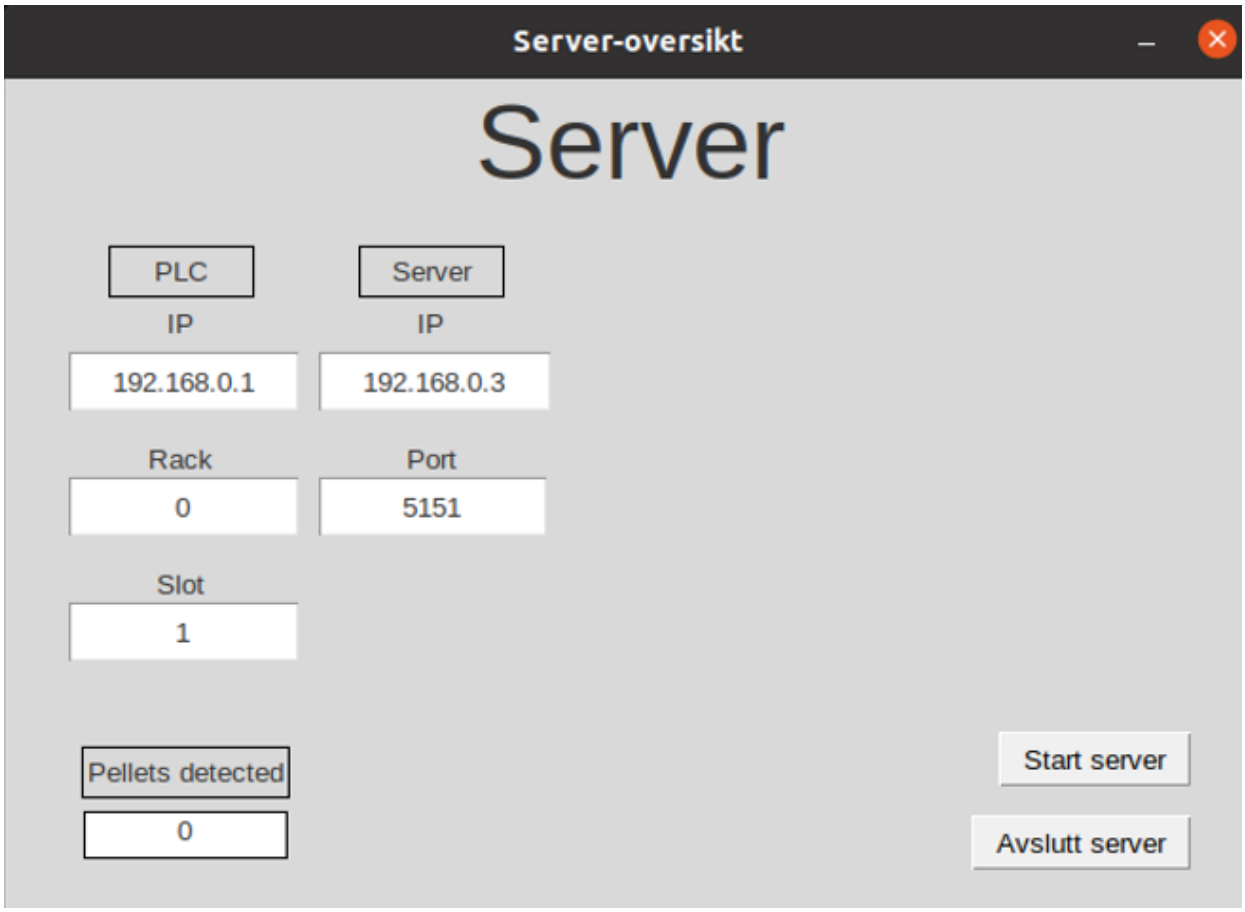
Figur 13 - Pascal-VOC

Detaljert arbeid beskrivelse om trening, testing og validering står i kapitelet «Bildedeteksjonsmodell».

5.1 Kode

Dette avsnittet beskriver kildekode som er programmert i oppgaven. Formålet med koden er å flette inn alle komponenter og programvare i oppgaven. Pythonskriptene styrer kommunikasjon mellom enhetene, og blir kjørt på KI-mikrokontroller. Koden behandler også objektdetektering, slik at PLS bare skulle motta enkle signaler som «start/stop» fôring.

5.1.1 ServerGUI.py



Figur 14- Server GUI

ServerGUI-programmet er kommunikasjons knutepunktet mellom Client.py på NVIDIA Jetson og PLS-en. Her er det bibliotekene Snap7 og Socket som blir brukt for kommunikasjon. Dette programmet ble programmert i IDE'n Visual Studio Code.

Det ble også laget et grafisk bruker grensesnitt, se Figur 14. GUI-en ble laget gjennom pythonbiblioteket Tkinter og viser en oversikt over IP adresser, som kan endres på ved behov. Hvis man har behov for å koble til en annen PLS har man mulighet til å endre IP-adresse i GUI-en uten å gå inn i kildekode.

«Pellets detected» tekstboksen i Figur 14 som skal vise antall pellets, er ikke tatt i bruk. Hele koden kan leses i <Vedlegg 2>.

5.1.2 Client.py

Client-programmet ble lagd ved å programmere inn en «launcher» av bildedeteksjonsmodellen ved hjelp av DetectNet biblioteket. I tillegg ble det satt begrensinger for når det skulle sendes beskjed om overfôring, og når fôringen skal stanses. Programmet styrer også hvor mange deteksjoner som skal til før det avgjør om fôring skal stanses. Client-programmet vil ikke kjøre dersom serveren ikke er startet i forkant. Dette er på grunn av Socket-kommunikasjon til serveren. For å se hele koden se <vedlegg 3>.

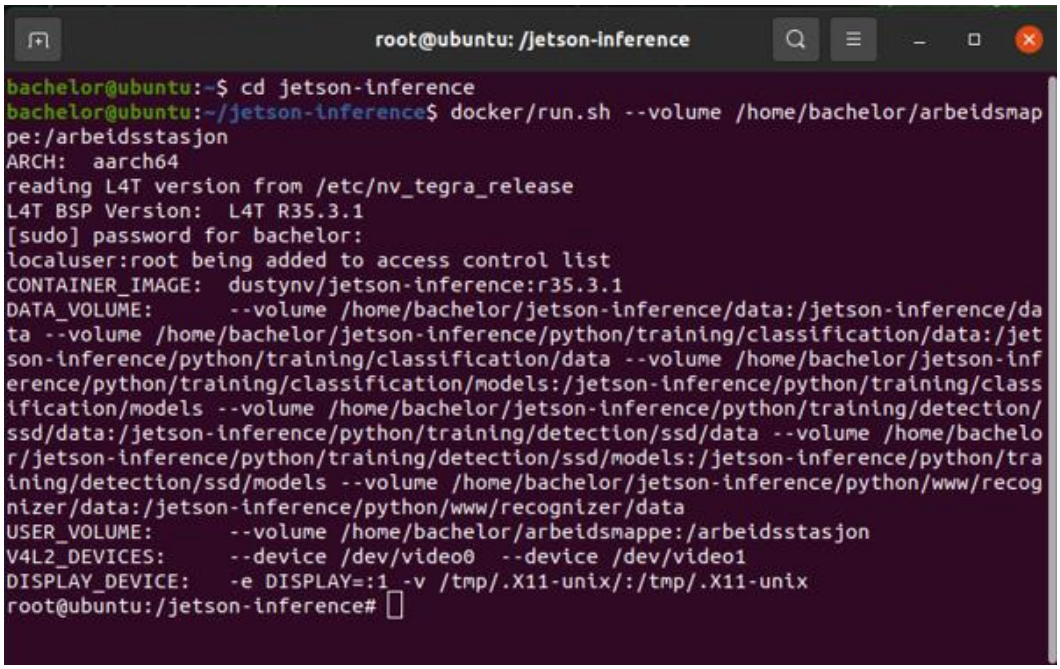
Client-programmet blir brukt til å behandle og starte objektdetekteringen før den sender en variabel til serveren.

5.1.3 PLS kode

PLS-koden består av en funksjonsblokk som styrer lys basert på om vribryter står i Automatisk eller Manuell. Koden er blitt programmert i «TIA Portal V16»

Dersom vribryter er i Automatisk vil lysstyring skje ved hjelp av «input» fra objektdetekteringen på KI-mikrokontrolleren. Inputen fra KI-mikrokontrolleren blir sendt til datablokken dbCommuncation og deretter brukt som input i funksjonsblokken. Dersom vri bryter står i manuell, vil PLS koden ignorere inputene fra KI-mikrokontroller og ikke styre lys påvirket av mikrokontrollerens signal. For å se hele koden se <Vedlegg 4>.

5.2 Docker – Jetson-Inference

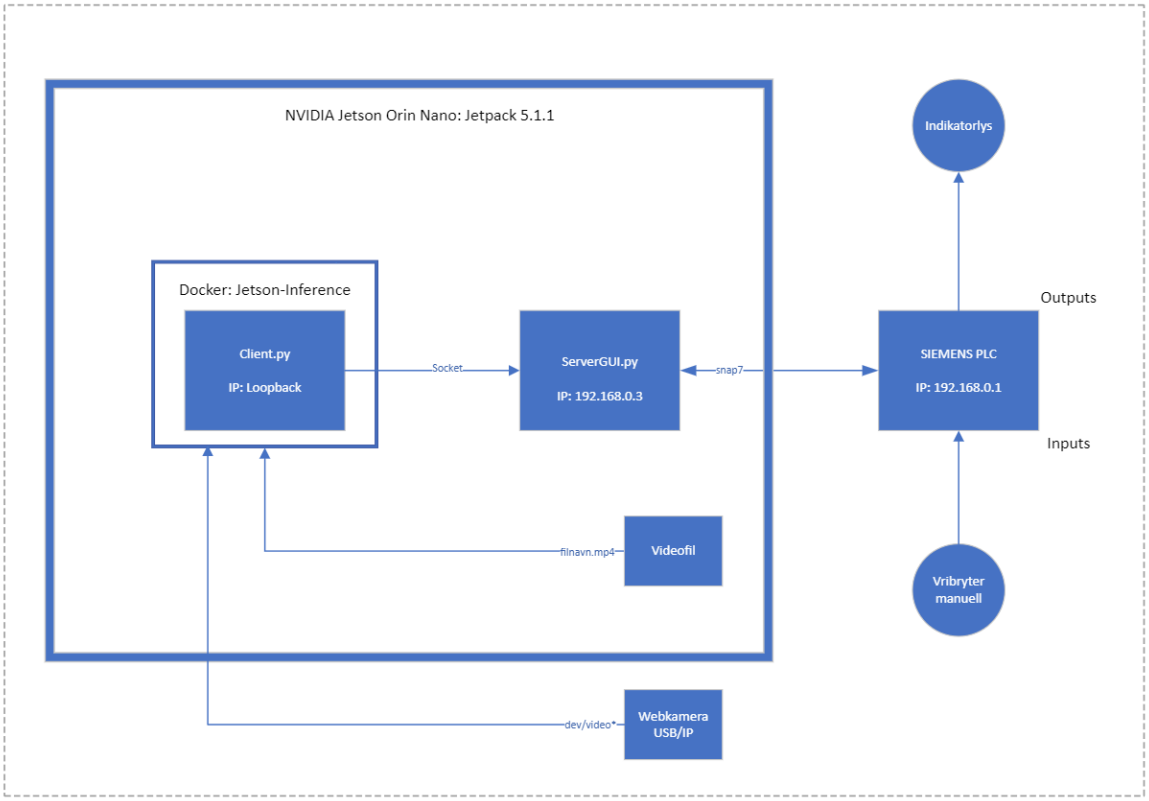


Figur 15- Jetson-Inference docker container

Figur 15 viser terminalen til dockercontaineren Jetson-Inference etter den er startet, det er i denne terminalen skriptet Client.py startes. Terminalen starter også treningen av datasettet med de ulike parameterne og konvertering fra PyTorch til ONNX.

En docker er et separat system som begrenser tilgang til alle mapper og filer i Linux operativsystemet. Docker har en kommando som heter «–volume». Funksjonen til «volume» er at den kan hente en mappe med filer som ikke er i dockeren. I prosjektet brukes det en mappe som er kalt «arbeidsmappe», hvor eksporterte objektdetekteringsmodeller og Client.py ligger lagret.

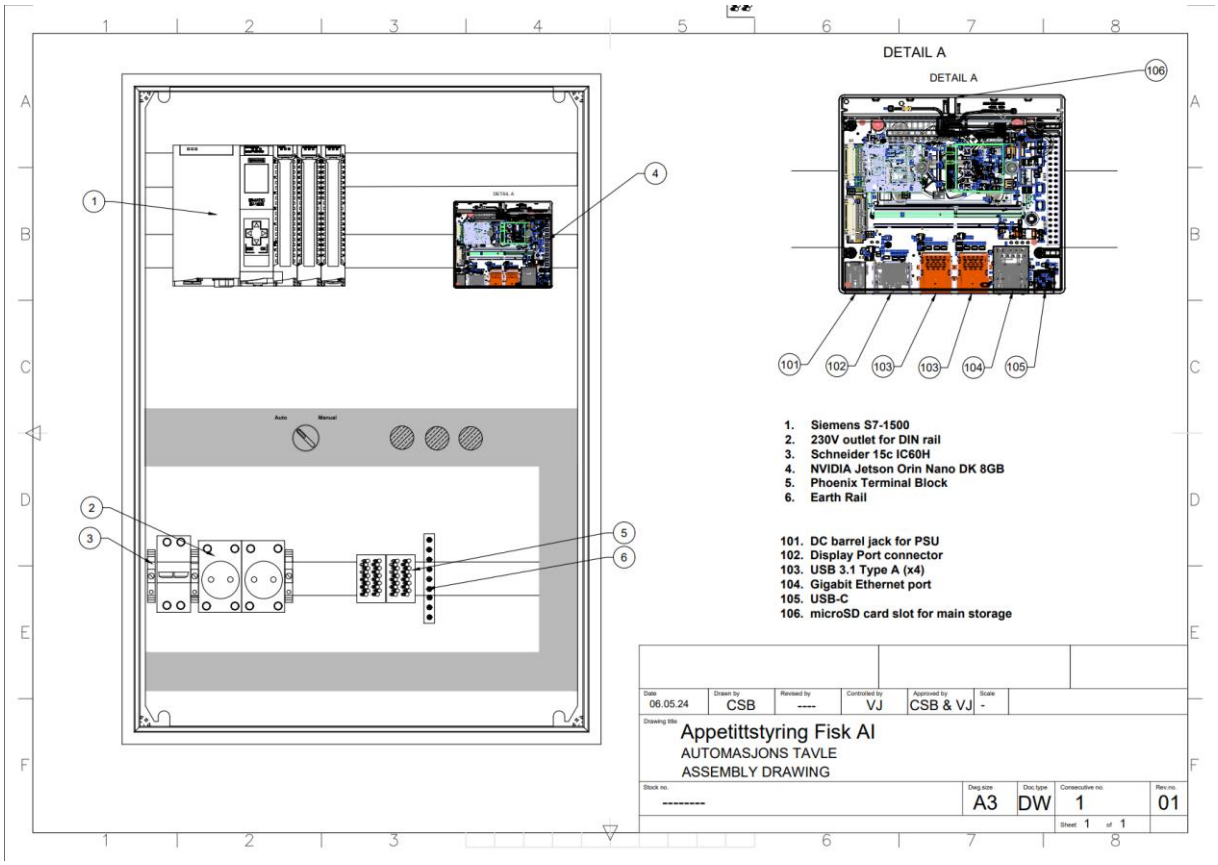
5.3 Beskrivelse av kommunikasjonsoppsettet



Figur 16 - System kommunikasjonsoversikt

Systemet bruker Ethernet for kommunikasjon, der ServerGUI.py har pythonbibliotekene Socket og Snap7 for kommunikasjon over Ethernet. PLS-en bruker Snap7, og mikrokontrolleren bruker Client.py programmet. Sistnevnte bruker Socket for kommunikasjon med server. Siemens S7-1512C-1 PN er en PLS med egen CPU og vil dermed kjøre uavhengig om serveren kjører. Client.py er avhengig av at serveren kjører før den vil starte programmet.

5.4 Automasjonstavle



Figur 17- AutoCAD tegning, Automasjonstavle

Automasjonstavle, se Figur 17**,** inneholder alt av maskinvare som blir brukt i oppgaven utenom skjerm, mus og tastatur. Tavlen ble laget for å gjøre det enklere å kunne jobbe med prosjektet på ulike lokasjoner og gjøre arbeidet mer oversiktlig. Det gjør det også mulig og ta tavlen til et ekte fiskeoppdrett for testing. Å slippe å koble opp PLS og mikrokontroller med tilleggsutstyr sparer tid når dette er fastmontert. Se koblingsskjema til tavlen i <Vedlegg 5>.

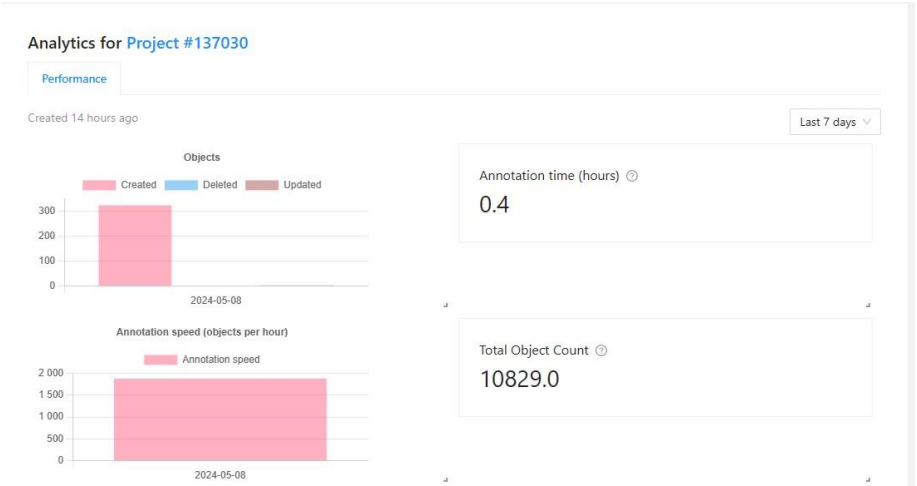
6 Bildedeteksjonsmodell

Kapittelet tar for seg trening og treningsresultater av bildedeteksjonsmodellene, samt hvordan tap i treningen og validering av modellen, og kan brukes til å indikere hvor god modellen er til og tilpasse seg nye objektet som trenes på. Videre vises også resultater av testing av de ulike bildedeteksjonsmodellene som har blitt trent.

6.1 Trening og resultat

Det er utarbeidet en egen bildedeteksjonsmodell i dockeren Jetson-Inference ved hjelp av PyTorch med en SSD300 v.1. Arkitekturen til SSD er illustrert i Figur 9. Utover dette blir modellen brukt til trening, og deretter eksportert dette til en ONNX modell. Alle funksjonene er implementert i Jetson-Inference. Grunnen til at modellen blir eksportert til ONNX er fordi det gjør det lettere for modellen å få tilgang til maskinvareoptimalisering. Eksporteringen gjøres, fordi systemet skal få mest mulig datakraft ut av KI-mikrokontrolleren og at modellen kan kjøres med TensorRT.

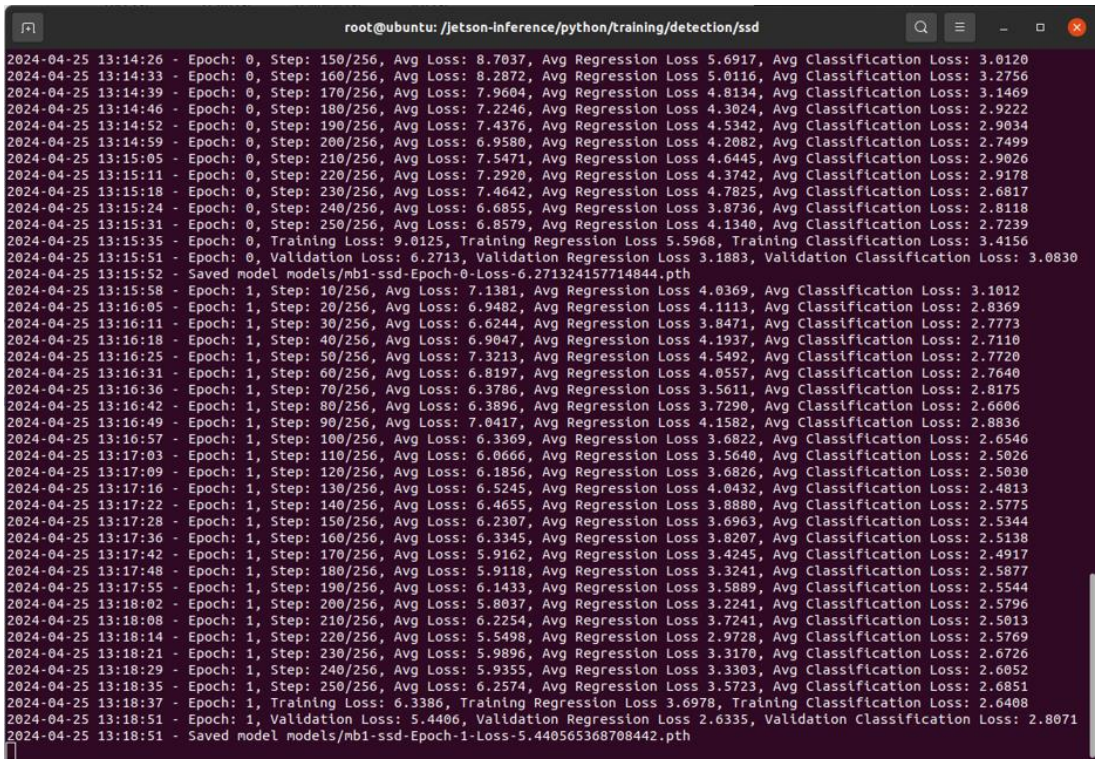
Arbeid før en kan starte å lage en modell er annotering av trening, test og valideringsdatasett. Dette ble lagd av gruppen ved hjelp av CVAT.ai. Framgangsmåte for å starte trening på Jetson-Inference finnes i <Vedlegg 1>.



Figur 18- Analyse CVAT

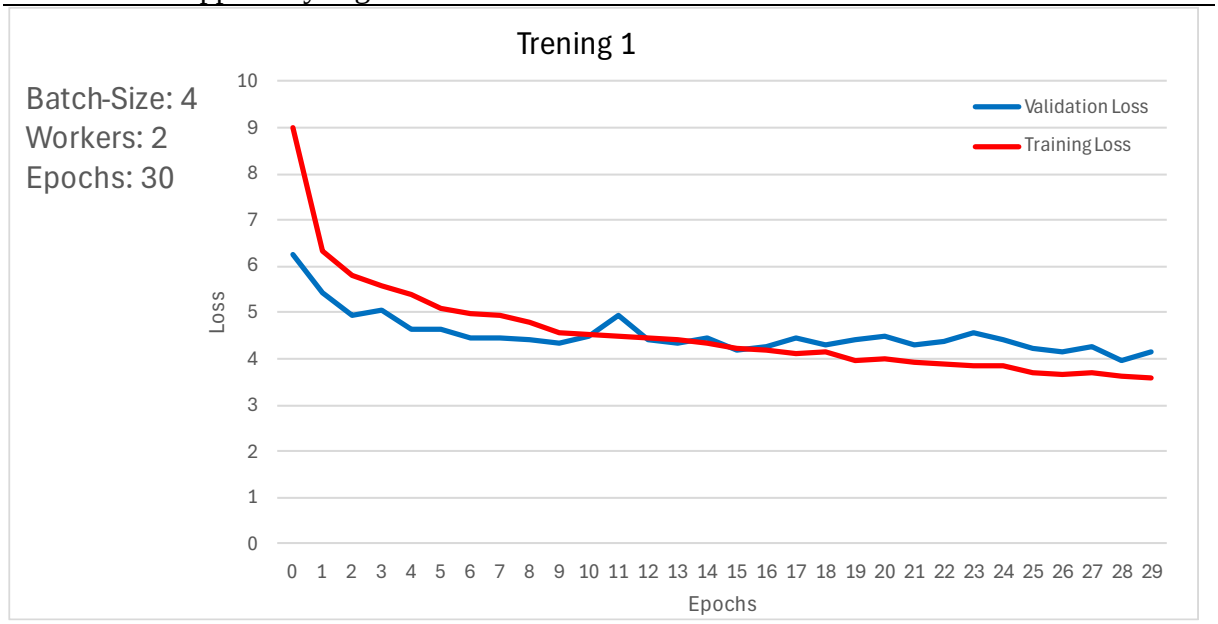
Omtrent 1500 bilder er fra 10 ulike videoklipp med forskjellige lysforhold og viskositet. Omtrent 80% er treningssett, 10% valideringssett og 10% Testingssett med totalt 10 829

objekter som vil gi omtrent 7 pellets per bilde, se «Figur 18». Siden det er såpass mange objekter per bilde gjør det at mengden bilder i datasettene kanskje er stor nok for trening. Om størrelsen og innholdet i datasettet har god kvalitet, kan tapet i trening og validering gi en indikasjon på en «good fit» modell. Se kapittel 2.3 og 2.4 for beskrivelse av begrepene.



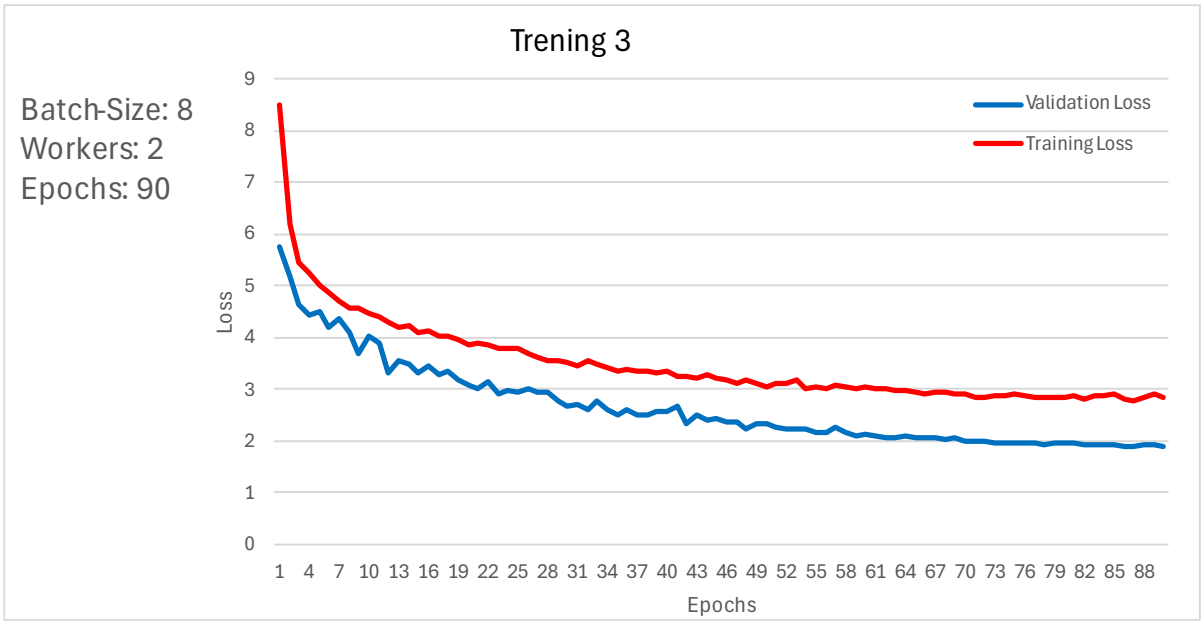
Figur 19 - Trening Jetson-Inference docker container

Treningen av datasettene ble utført flere ganger med ulike parameter, «Batch-size», «workers» og «epochs». «Batch-size» er antall treningsprøver som behandles før modellen oppdateres. «Workers» er antall parallelle prosesser eller tråder som brukes for å laste data. «Epochs» er hvor mange ganger datasettene trenes, og etter hver «epoch» sendes en melding i terminalen der det står verdi på tap av validering og trening, se Figur 19. Som beskrevet i kapittel 2.4, så er verdier nærmest mulig 0 optimalt. Treningen kan logges ved å skrive ned disse verdiene og plotte dem. Mer forklaring om tap i validering og trening står i kapittel 2.4.



Figur 20- Plot trening 1

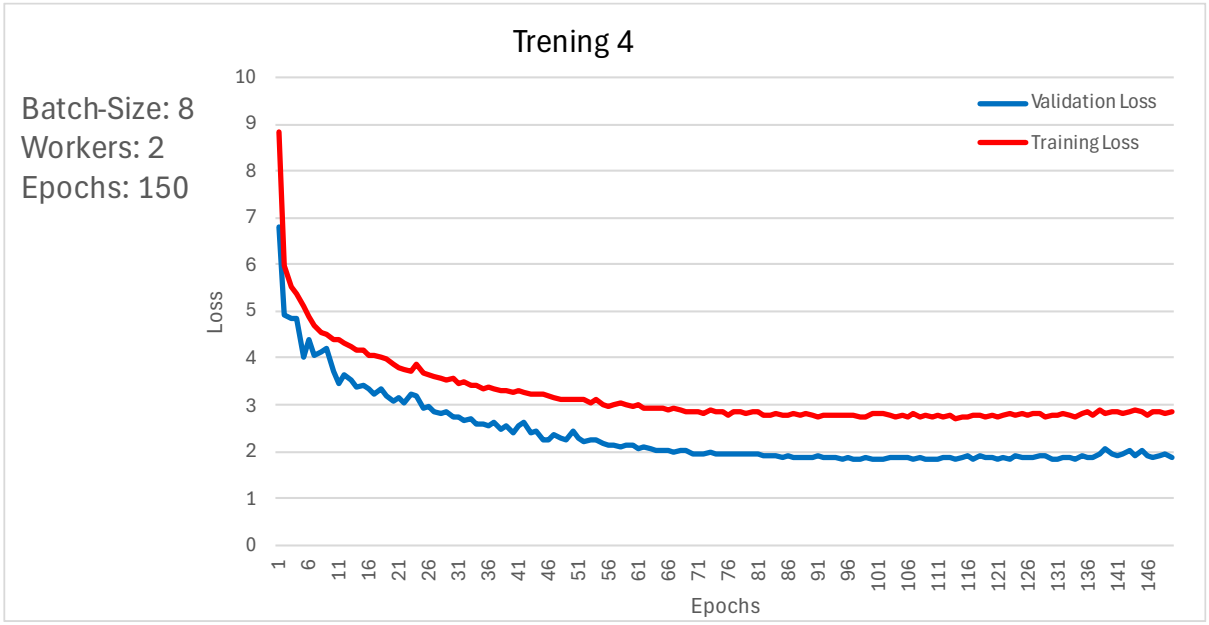
Figur 20 viser en «overfit», fordi treningstapet krysser valideringstapet, og valideringstapet flater ut. Det betyr at den har lært trening datasettet for godt som gjorde at det ble lagt til flere bilder i treningssettet.



Figur 21- Plot trening 3

Figur 21 viser validering og treningstap fra trening 3 av modellen. Det ble ikke en «overfit» på denne slik som i Figur 20, men det ble heller ikke noe tydelig «good fit». Tap på trening og validering er i underkant av 3 på trening og underkant 2 på validering. Modellen ble testet

på testdatasett fordi grafen viser ikke til et tydelig resultat. Det ble hentet ut noen statiske bilder for å sammenligne med tidligere treninger. Ut ifra sammenligningen var det forbedring. Dette førte videre til at en ytterligere trening av datasettet ble utført. Denne gangen med 150 «epochs» for å forsøke og forbedre modellene enda mer. Se <Vedlegg 6> for de treningene som ble loggført.



Figur 22- Plot trening 4

Figur 22 viser trening og validerings tap fra trening 4, dette er modellen som blir brukt i oppgaven. Resultatet fra dette plottet av tap i trening og valideringen ser vesentlig likt ut som trening 3 i Figur 21. Modellen har ikke en tydelig «good fit», men ut ifra testene som er gjennomført i «6.2» kan modellen brukes til prosjektets formål.

6.2 Testing av Bildedeteksjonsmodell

Figur 23 – Testbilde trening 1. hentet fra https://www.ewos.com/ca/products-and-services/salmon/

Figur 23 viser et bilde som ble brukt til å teste bildedeteksjonsmodell fra trening 1. Bildet har ikke like lysforhold eller noen sammenheng med noe av trening eller validerings datasettene. De ulike lysforholdene er for å gi en ekstra utfordring i testingen av modellen. I testbilde er det mange pellets, men ikke mer enn 4 deteksjoner. Deteksjonene i bilde treffer heller ikke noen pellets, men har ganske god sikkerhet på at det er pellet den detekterer. Deteksjonene er i nærheten av pelletene, men ikke bra nok. Det virker som den detektere, men klarer ikke å plassere «bounding boxene» i riktig posisjon. «Localization error» kan ha noe med at objektene er såpass små, samt oppløsningen på bilde kan også spille en rolle.

Recall og presisjon blir forklart i 2.5.

Recall og Presisjon på deteksjoner i Figur 23**:**

$$\text{Recall} = \frac{\text{Ekte Positiv}}{\text{Ekte Positiv+Falske Negative}} = \frac{1}{1+42} = 0.023 = 2.3\%$$

Recall gir en prosent på hvor mange pellets som faktisk blir detektert ut ifra den totale mengden pellets i bilde.

$$\text{Presisjon} = \frac{\text{Ekte Positiv}}{\text{Ekte Positiv+Falske Positiv}} = \frac{1}{1+3} = 0.25 = 25\%$$

Presisjonen er i realiteten ikke på 25%, fordi den ene ekte positive identifiseringen som blir brukt i formelen har stor «localization error». For dette prosjektet er det viktigste deteksjoner av pellets, men ikke nødvendigvis plasseringen på hvor pelletene befinner seg.

Evaluering etter denne testen er at modellen var åpenbart for dårlig, og har for lite deteksjoner og treffsikkerhet.

BO24EH-02 Appetittstyring Fisk KI

Test 2 og 3 har litt forbedring, men ikke nok til at det var nødvendig å gjør utgreiing om de.

Figur 24 – Testbilde trening 4. hentet fra https://www.ewos.com/ca/products-and-services/salmon/

I Figur 24 er det en god del deteksjoner, så det har vært tydelig fremgang i treningen som er utført.

$$\text{Recall} = \frac{\text{Ekte Positiv}}{\text{Ekte Positiv+Falske Negative}} = \frac{9}{9+34} = 0.209 = 20.9\%$$

Etter de ulike justeringer i treningen av datasettene viser Figur 24 i utregningen av «recall» at modellen er forbedret med 18.6 prosent, modellen er fortsatt ikke helt optimal, men det kan være nok for oppgavens formål.

$$\text{Presisjon} = \frac{\text{Ekte Positiv}}{\text{Ekte Positiv+Falske Positiv}} = \frac{9}{9+6} = 0.6 = 60\%$$

Presisjonen i Figur 24 er også en hel del bedre enn det var etter første trening i Figur 23.

Det ble testet en del mer på testdatasettet og det ble evaluert med at modellen kan brukes til denne oppgaven.

Figur 25- Testbilde 2, trening 4.

Figur 25 viser ett testbilde fra testdatasettet der lysforhold og viskositet er mer likt lysforhold og viskositet i treningssettet. Dette viser også igjen på resultatene på deteksjonene.

$$\text{Recall} = \frac{\text{Ekte Positiv}}{\text{Ekte Positiv+Falske Negative}} = \frac{5}{5+1} = 0.833 = 83.3\%$$

Her er det bra recall og modellen detekterer nesten alle pellets i test bilde.

$$\text{Presisjon} = \frac{\text{Ekte Positiv}}{\text{Ekte Positiv+Falske Positiv}} = \frac{5}{5+0} = 1.0 = 100\%$$

Presisjonen i dette test bilde er bra, men pelleten til venstre i figur 24 har noe «localization error».

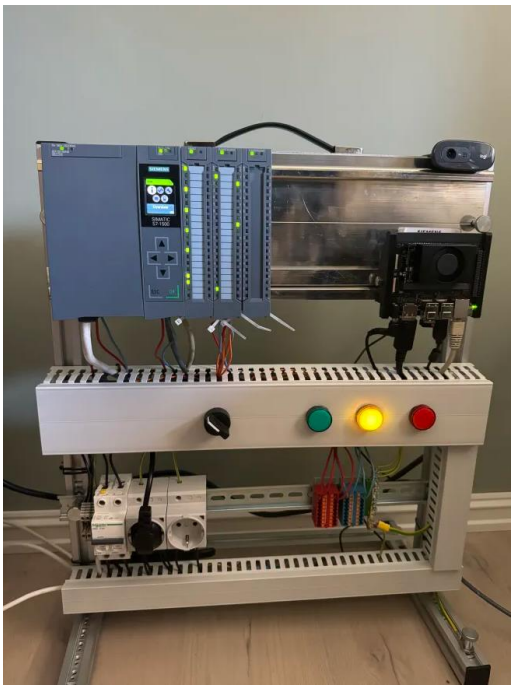
Samlede resultater:

Trening nr.	Recall	Presisjon
Trening 1	2.3%	25 %
Trening 4 -Bilde 1	20,9 %	60 %
Trening 4 -Bilde 2	83.3%	100 %

7 Systemtesting

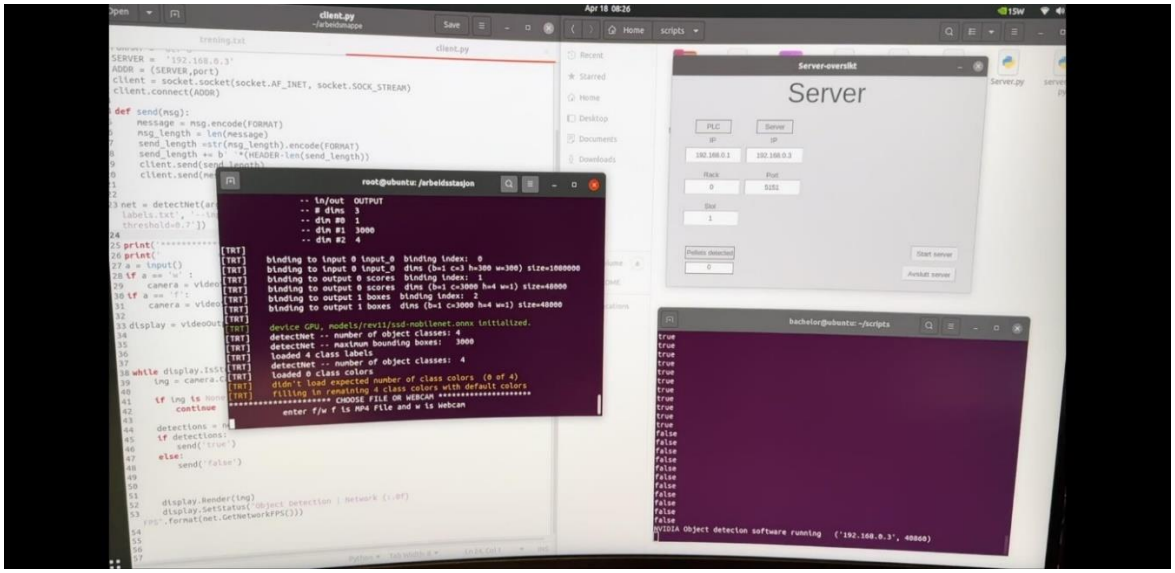
I dette kapittelet blir hele systemet testet for å se at alle komponenter kommuniserer, og at løsningen får ønsket resultat. Ønsket resultat med testingen er at når systemet er satt i automatisk styring, skal systemet kjøre «fôring pågår» tilstand (grønt lys), se Figur 26. Ved stor mengde pellets skal PLS-en endre tilstand til «pause fôring», og gi gult lys, se Figur 27. Dette baseres på deteksjoner som KI-mikrokontrolleren gjør, og sender verdiene til PLS. Dersom systemet står i manuell, vil PLS ignorere meldinger fra KI-mikrokontroller.

Figur 26 - Automasjonstavle tilstand «fôring pågår»



Figur 27- Automasjonstavle tilstand «fôring pause»

Hele systemet ble testet med flere videoer med et stort antall pellets, for å teste at systemet gikk i tilstand “fôring pauset” når det var mye pellets i bilde.



Figur 28 - Testing av system med videofil

I Figur 28 er det mulig å se i terminalen nede til høyre med de boolske verdiene som blir sendt til serveren fra objektdetekteringen, basert på mengde pellets. Systemet fungerer som det skal i alle videoer under denne testingen, men systemet kunne vært noe mer presist i selve objektdetekteringen.

Under den andre testingen ble det forsøkt å fjerne nettverkskabel fra PLS. Ved fôring var PLS-en fortsatt i tilstand «fôring pågår». En annen test som ble utført var dersom en av programkodene på mikrokontrolleren ble avsluttet, skifter ikke tilstanden fra «fôring pågår» til «fôring pause». Det vil si at dersom problemer oppstår med systemet, så vil fôring fortsette uten stans. Dette er kritiske funksjoner som burde blitt implementert i PLS-kode.

8 Diskusjon

I dette kapittelet blir de ulike utfordringene som ble møtt i løpet av prosjektet forklart. Kapittelet forklarer også rammene for den opprinnelige fremdriftsplanen som ble satt i forprosjektet, se Figur 30. Underveis i prosjektet ble det møtt på utfordringer som har tatt lenger tid i forhold til fremdriftsplan, og dermed gitt forskyvninger i planen. Samtidig har det også vært oppgaver som har gått raskere enn planlagt, som har gjort at det er utført små endringer i fremdriftsplanen underveis.

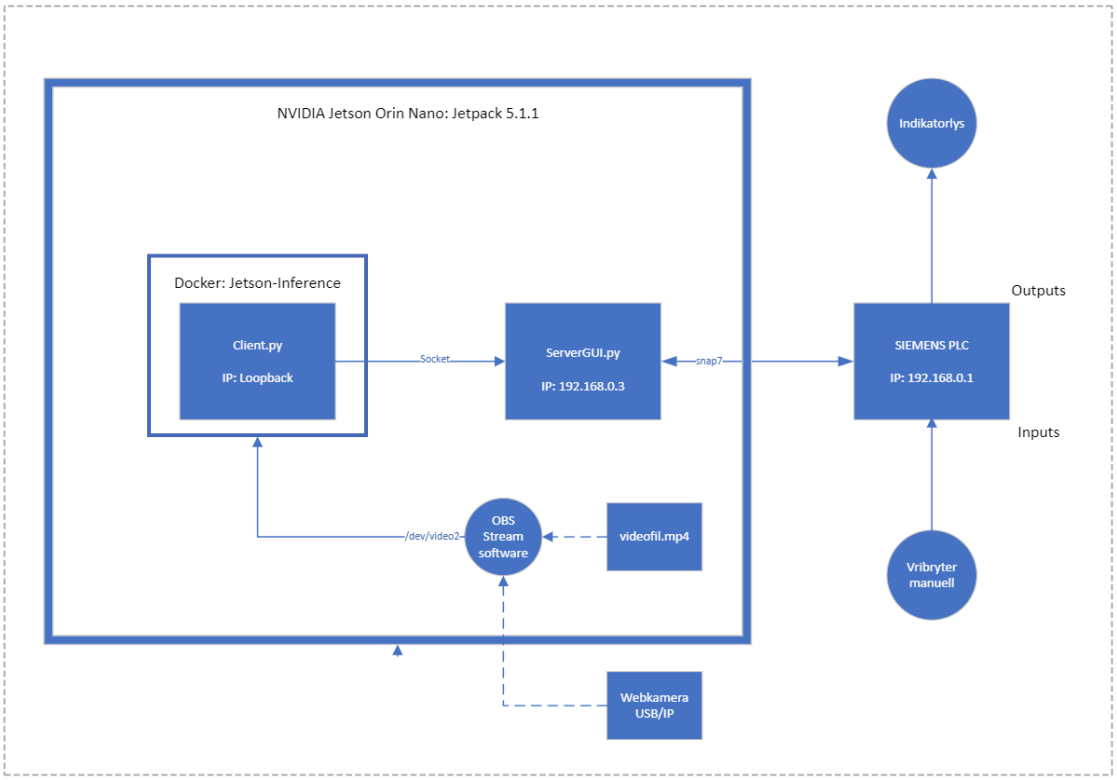
8.1 Utfordringer under oppsett av system

Installering av operativsystem:

Det ble først prøvd å «flashe» operativsystem «image» med SD kort. Når mikrokontrolleren ble forsøkt startet, kom det feilmelding ved oppstart. Problemet hvis man prøver å installere operativsystem med SD kort, hvor operativsystemet ikke stemmer overens med versjonen på QSPI-modulen, vil ikke dette fungere. Løsningen er å installere ved bruk av SDK Manager, og da oppdateres QSPI-modulen samtidig.

Kommunikasjon:

- Ingen meldinger fra KI-mikrokontroller ble mottatt av PLS, det måtte derfor endres innstillinger på PLS for å gjøre det kompatibelt med TCP/IP gjennom Snap7-biblioteket på mikrokontroller. Se <Vedlegg 7> for å se oppsett av PLS.
- Docker – Jetson-Inference på nettverk kommunikasjon. Prøvde først å definere en IP-adresse for Docker containeren. Det viste seg at Docker er innebygd med en loopback IP-adresse. For å se systemets kommunikasjon oversikt, se Figur 16.



Figur 29 - Opprinnelig plan for systemkommunikasjon

I oppstarten ved trening av objekter med bilder ble det brukt treningsprogramvaren til Jetson-Inference, Camera-Capture. Der ble det møtt et par utfordringer med videostrømming, fordi at programvaren bare var kompatibel med videostrømming over IP, eller webkamera. Det ble gjort et forsøk på å strømme videofiler via OBS. Figur 29 viser opprinnelig plan for systemkommunikasjon der OBS var inkludert i oversikten. OBS var derimot ikke støttet på mikrokontrollerens operativsystem. Grunnet disse utfordringene ble CVAT.AI brukt som annoteringsverktøy, og deretter ble datasettene eksportert til mikrokontrolleren.

8.2 Utfordringer rundt videoopptak

Det var en del utfordringer rundt videoopptakene. Noen av opptakene som ble undersøkt om de kunne brukes til oppgaven, inneholdt få pellets eller små pellets. Videoene som har få/små pellets ble ikke brukt i datasettene. Størrelsen på video oppløsningen kunne også være et problem. Videoene som ble brukt til trening og testing burde hatt samme størrelse på oppløsningen, eller enda større variasjon i oppløsningen. Imidlertid ville dette krevd et større datasett.

8.3 Nøyaktighet ved deteksjon

Når det gjelder nøyaktighet ved deteksjon er ingen metode perfekt, det vil som regel alltid være noe usikkerhet knyttet til objektdeteksjon. Derfor ble det testet mye for å forbedre nøyaktigheten i deteksjonene. I oppgaven var det ikke god nøyaktighet i alle testvideoer og testbilder på deteksjon av pellets. Det er fortsatt mye arbeid som kan bli gjort for å forbedre modellen ytterligere. Noe som kan lage forstyrrelser på nøyaktigheten ved deteksjon er for eksempel størrelse på formatet til videoen/bildene og hvor god oppløsningen er.

8.4 PLS – Fôringsstyring basert på inndata

PLS'en får inndata fra ServerGUI.py på KI-mikrokontrolleren, og denne sender et tall som settes til boolske verdier i PLS programmet. Det avgjør hvilken tilstand PLS'en blir satt til, om det er «pause fôring» eller «start fôring» som pågår. Dette er noe som kunne blitt gjort på en mer omfattende måte, der det ble sendt inn antall pellets og regnet ut estimat på overfôring i kilo. Kravet til dette er at bildegjenkjenningsmodellen er god, for at systemet skal gi nøyaktige resultater.

8.5 Automatisk styring kontra Manuell styring

Automatisk styring gir en fordel til fiskeoppdrett lokaliteten, da det kan frigjøre en operatør til å ha mulighet til å gjøre andre viktige oppgaver på oppdrettsanlegget. En automatisk fôring jobber kontinuerlig, men med manuell styring er risikoen for overfôring større, dersom operatøren mister oppmerksomhet.
Ulempene med Automatisk styring er at det burde tatt hensyn til flere faktorer som operatører kan evaluere, men som ikke blir evaluert i den automatiske løsningen. Bevegelse mønster på fisken, fiskens aktivitet og mengde pellets som blir tatt av strøm osv. Dette er noe som for eksempel kan løses med automatisk styring med KI som analyserer fiskens atferd.

Derfor har løsningen blitt laget slik at systemet styres av en vribryter, enten settes PLS'en i Automatisk eller Manuell.

9 Konklusjon

I resultatet fra treningen ble det noen utydelige resultat ut fra trening og valideringstap. Det var noe høyt tap per trening, som gjør at modellen i løsningen ikke har en tydelig «good fit» ut fra grafene. «Good fit» vil si at tap på både validering og trening har tilnærmet lik tall verdi. Ønsket tap på trening og valideringer på en plass mellom 0 og 1, men modellen som ble laget i prosjektet hadde et tap på ca. 2 på validering og 3 på trening.

Derimot når bildedeteksjonsmodellen ble testet, var resultat bedre enn trening og valideringstapet indikerte. Resultatet fra den siste modellen hadde recall gjennomsnitt på 52.1% og presisjon på 80%. Det er fortsatt «localization error» på modellen. Konklusjonen er at modellen detekterer pellets, men den burde hatt enda høyere recall for enda bedre estimering av overfôring.

For å optimalisere modellen enda mer burde det bli brukt enda flere videoopptak med enda flere lysforhold og viskositeter.

I resultatet fra Systemtesting burde det blitt implementert sikkerhetsfunksjoner i PLS-programmet, dersom den mistet kommunikasjon med KI-mikrokontroller. Dette er kritisk i fôringsprosessen, resultatet kan i ekstreme tilfeller ende i overfôring dersom PLS blir stående i tilstand «fôring pågår» uten kommunikasjon med KI-mikrokontroller. Uten kommunikasjons brudd fungerer systemet tilfredsstillende.

Systemet viser et potensial for bruk ikke bare innen oppdrettsindustrien, men også i en rekke andre områder. I tillegg viser systemet å ha en god stabilitet mellom KI-mikrokontrolleren og PLS. Det gir mulighet for å regulere og styre prosesser dynamisk basert på direkte informasjon fra en videostrøm.

9.1 Mulige utvidelser av løsningen

I oppgaven finnes det flere muligheter for utvidelser både på objektdeteksjonen, og på styringen av systemet. Her er noen av utvidelsene som kan være fornuftige å vurdere.

- Forbedring av bildedeteksjonsmodell, enda mer trening, testing og validering med større datasett som gir enda mer nøyaktighet ved deteksjon.
- Implementering av sikkerhetsfunksjoner i PLS programmet ved kommunikasjonsbrudd.
- Undersøke «small object detection», og om det finnes algoritmer som egner seg bedre for små objekter.
- Undersøke muligheten ved å lage en modell som bruker «object tracking», slik at den kan bli brukt til å identifisere fiskeatferd. Dette kan gi indikasjoner om fisken er aktiv når den blir foret eller om den er «sløv» og inaktiv, noe som kan potensielt være en mer presis indikator på appetittstyring.
- Se på mulighetene for en KI-assistent som tar vurderinger istedenfor logikk i programkode basert på deteksjoner. Her kan nylig lanserte NVIDIA JetPack 6.0 og Llava ha potensiale.
- Prøve å implementere systemet i et reelt PLS-fôringsprogram og teste på en lokasjon istedenfor igjennom videoopptak.
- Bruke systemet til å detektere andre objekter for å styre en annen prosess enn i dette prosjektet. For eksempel et parkeringssystem.

Referanser

[1] Helgevold Gruppen. «Om oss» Helgevold. Hentet fra: https://helgevold.com/helgevoldgruppen/ (Lastet ned: 20.02.2024)

[2] M. Espinasse, E. Mikkelsen, & P. Fauchald, «Forurensing fra lakseoppdrett» Kystbarometeret. Hentet fra: https://kystbarometeret.no/indikatorer/matproduksjon-akvakultur/artikkel/forurensing-fra-lakseoppdrett (Lastet ned: 20.02.2024).

[3] T. Stensvold, «Rapport: For å få ned klimaavtrykket til oppdrettslaks må fiskens spisevaner endres» Teknisk Ukeblad. Hentet fra: https://www.tu.no/artikler/rapport-for-a-fa-ned-klimaavtrykket-til-oppdrettslaks-ma-fiskens-spisevaner-endres/485344 (Lastet ned: 15.03.2024).

[4] A. Tidemann, «Kunstig intelligens» Store Norske Leksikon. Hentet fra: https://snl.no/kunstig_intelligens (Lastet ned: 25.04.2024).

[5] MathWorks, «What is Object Detection?» se.mathworks.com. Hentet fra: https://se.mathworks.com/discovery/object-detection.html#how-it-works (Lastet ned: 13.05.2024)

[6] J. Murel Ph.D, Eda Kavlakoglu, «What is Object Detection?» ibm.com. Hentet fra: https://www.ibm.com/topics/object-detection (Lastet ned: 14.05.2024)

[7] M. Olafenwa, «Object Detection Training – Preparing your custom dataset» medium.com. Hentet fra: https://medium.com/deepquestai/object-detection-training-preparing-your-custom-dataset-6248679f0d1d (Lastet ned: 16.05.2024)

[8] T. Shah, «About Train, Validation and Test Sets in Machine Learning» towardsdatascience.com. Hentet fra: https://towardsdatascience.com/train-validation-and-test-sets-72cb40cba9e7 (Lastet ned: 16.05.2024)

[9] J. Brownlee Ph.D, «How to use Learning Curves to Diagnose Machine Learning Model Performance» machinelearningmastery.com Hentet fra: https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-model-performance/ (Lastet ned: 11.05.2024)

[10] Google, «Classification: Precision and Recall» developers.google.com Hentet fra: https://developers.google.com/machine-learning/crash-course/classification/precision-and-recall (Lastet ned: 11.05.2024)

[11] Martin Aune Walther, Trine Merethe Paulsen. Lerøy, Norge. Fôring i sjøanlegget. (31.10.2017). Sett: 02. 20, 2024. [Online video]. https://ndla.no/subject:169ba831-b3cd-4207-b9b8-7d06bf03328b/topic:0f9ad778-a98b-4324-b99a-bfa23fd25221/topic:f9a97f87-db34-4352-9f7f-9282f062635b/resource:1:164525

[12] T. M. Paulsen, E. Skoglund, «Fôring av fisk» ndla.no. Hentet fra: https://ndla.no/subject:169ba831-b3cd-4207-b9b8-7d06bf03328b/topic:0f9ad778-a98b-4324-b99a-bfa23fd25221/topic:f9a97f87-db34-4352-9f7f-9282f062635b/resource:1:162587 (Lastet ned: 20.02.2024)

[13] NVIDIA Corporation, «Jetson Orin Nano Developer Kit Getting Started Guide. » developer.nvidia.com. Hentet fra: https://developer.nvidia.com/embedded/learn/get-started-jetson-orin-nano-devkit (Lastet ned: 02.02.2024).

[14] NVIDIA Corporation, «JetPack SDK 5.1.1» developer.nvidia.com. Hentet fra: https://developer.nvidia.com/embedded/jetpack-sdk-511 (Lastet ned: 02.02.2024).

[15] NVIDIA Corporation, «NVIDIA TensorRT» developer.nvidia.com. Hentet fra: https://developer.nvidia.com/tensorrt (Lastet ned: 01.05.2024).

[16] NVIDIA Corporation, «Deep Learning Frameworks» developer.nvidia.com. Hentet fra:

https://developer.nvidia.com/deep-learning-frameworks
(Lastet ned: 01.05.2024).

[17] NVIDIA Corporation, «SDK Manager» developer.nvidia.com. Hentet fra: https://developer.nvidia.com/sdk-manager
(Lastet ned: 20.02.2024).

[18] D. Franklin, «jetson-inference» github.com. Hentet fra: https://github.com/dusty-nv/jetson-inference
(Lastet ned: 01.03.2024).

[19] Docker Inc, «Docker overview» docs.docker.com. Hentet fra: https://docs.docker.com/get-started/overview/
(Lastet ned: 11.05.2024).

[20] Siemens AG, «6ES7512-1CK01-0AB0» mall.industry.siemens.com. Hentet fra: https://mall.industry.siemens.com/mall/en/WW/Catalog/Product/6ES7512-1CK01-0AB0
(Lastet ned: 11.05.2024).

[21] Siemens AG, «6ES7507-0RA00-0AB0» mall.industry.siemens.com. Hentet fra: https://mall.industry.siemens.com/mall/en/WW/Catalog/Product/6ES7507-0RA00-0AB0
(Lastet ned: 11.05.2024).

[22] Siemens AG, «Delivery release TIA Portal V16» support.industry.siemens.com. Hentet fra: https://support.industry.siemens.com/cs/document/109771626/delivery-release-tia-portal-v16?dti=0&lc=en-AZ
(Lastet ned: 11.05.2024).

[23] Python Software Foundation, «About Python» python.org. Hentet fra: https://www.python.org/about/
(Lastet ned: 11.05.2024).

[24] Python Software Foundation, «Installing Python Modules» docs.python.org. Hentet fra: https://docs.python.org/3/installing/index.html
(Lastet ned: 11.05.2024).

[25] D. Nardella, «Overview» snap7.sourceforge.net. Hentet fra: https://snap7.sourceforge.net/ (Lastet ned: 10.03.2024).

[26] Python Software Foundation, «tkinter – Python interface to Tcl/Tk» docs.python.org. Hentet fra: https://docs.python.org/3/library/tkinter.html (Lastet ned: 10.03.2024).

[27] D. Franklin, «jetson.inference» rawgit.com. Hentet fra: https://rawgit.com/dusty-nv/jetson-inference/master/docs/html/python/jetson.inference.html#detectNet (Lastet ned: 11.05.2024).

[28] Python Software Foundation, «socket – Low-level networking interface» docs.python.org. Hentet fra: https://docs.python.org/3/library/socket.html (Lastet ned: 10.03.2024).

[29] Cornell University, «SSD: Single Shot Multibox Detector» arxiv.org. Hentet fra: https://arxiv.org/abs/1512.02325 (Lastet ned: 11.05.2024).

[30] D. Franklin, «pytorch-ssd/train_ssd.py» github.com. Hentet fra: https://github.com/dusty-nv/pytorch-ssd/blob/6accaa88845ec135a7d6fe25e9a26afd4698639d/train_ssd.py (Lastet ned: 01.03.2024).

[31] PyTorch Contributors, «PyTorch documentation» pytorch.org. Hentet fra: https://pytorch.org/docs/stable/index.html (Lastet ned: 10.05.2024).

[32] ONNX, «ONNX documentation» onnx.ai. Hentet fra: https://onnx.ai/onnx/index.html (Lastet ned: 10.05.2024).

[33] D. Franklin, «pytorch-ssd/onnx_export.py» github.com. Hentet fra: https://github.com/dusty-nv/pytorch-ssd/blob/6accaa88845ec135a7d6fe25e9a26afd4698639d/onnx_export.py (Lastet ned: 10.05.2024).

[34] Microsoft Corporation «Visual Studio Code Docs» visualstudio.com. Hentet fra: https://code.visualstudio.com/docs (Lastet ned: 15.05.2024).

[35] Github Inc, «About GitHub and Git» docs.github.com. Hentet fra: https://docs.github.com/en/get-started/start-your-journey/about-github-and-git (Lastet ned: 15.04.2024).

[36] CVAT.ai Corporation ,«CVAT Overview» docs.cvat.ai. Hentet fra: https://docs.cvat.ai/docs/getting_started/overview/ (Lastet ned: 10.05.2024).

Appendiks A Prosjektledelse og styring

A.1 Prosjektorganisasjon

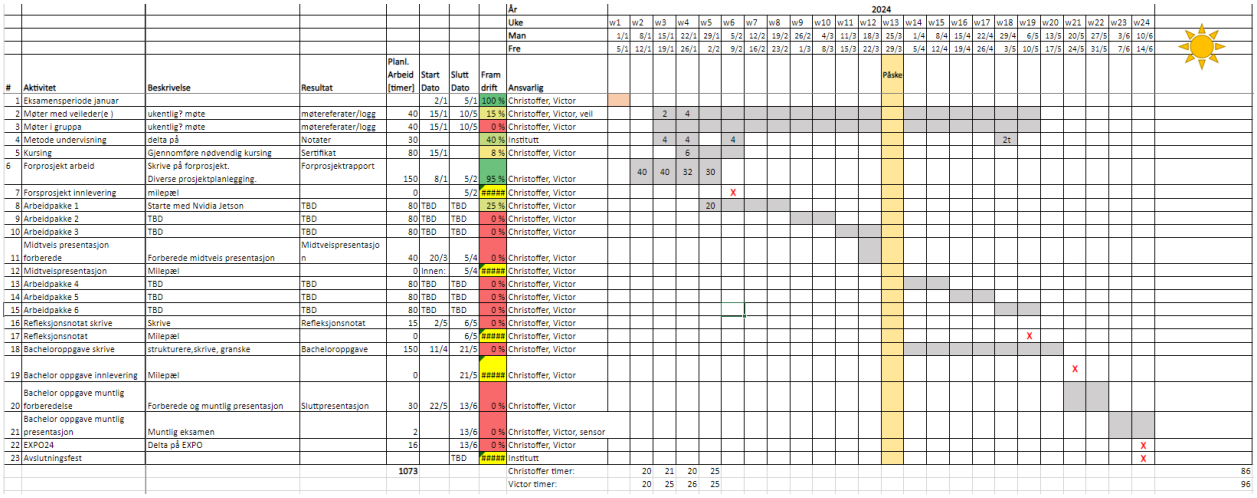
Prosjektorganisasjonen består av to deltakere, hvor Victor ble valgt som prosjektleder på grunn av hans arbeidsforhold i samarbeidsbedriften, som gjorde det enklere for han å håndtere kommunikasjonen mot bedriften og mellom veiledere. Utover dette ble ansvaret til prosjektet delt helt likt. Begge deltakerne hadde like stor innflytelse på alle beslutninger som ble tatt, og ansvaret ble jevnt fordelt. Samarbeidet fungerte svært godt, og arbeidsfordelingen føltes naturlig. Grunnet valget ved å være en gruppe på to medlemmer ble det effektivt å samarbeide på samme deler av prosjektet under utviklingsstadiet, og deretter jobbe med hver vår oppgave, lufte tanker og spørsmål underveis.

A.2 Prosjektform

Gruppen valgte å implementere en egendefinert prosjektform som startet med utviklingen av et Gantt-diagram i forprosjektet. Dette diagrammet ga en overordnet oversikt over prosjektets tidsplan og arbeidspakker. Underveis innså gruppen for å prioritere sammensetningen av det totale systemet i prosjektet, før det ble investert betydelig med tid i bildebehandlingstrening. Dette ledet til en justering av planen hvor systemet ble delt opp i forskjellige delmål, og milepæler ble satt for hvert delmål som skulle oppnås. Alt av filer og arbeidsdokumenter ble delt i Microsoft Teams. I Teams benyttet gruppen seg også aktivt av oppgaveplanleggeren for å legge inn frister for innleveringer og presentasjoner, samt dele oppgaver med hverandre.

Gjennom denne måten å jobbe på opplevde gruppen jevnlig mestringsfølelse når delmålene ble nådd, noe som motiverte underveis. Ved å tilpasse prosjektets form etter behov og rådgivning fra veiledere kunne gruppen oppnå en effektiv gjennomføring og oppnå ønskede resultater.

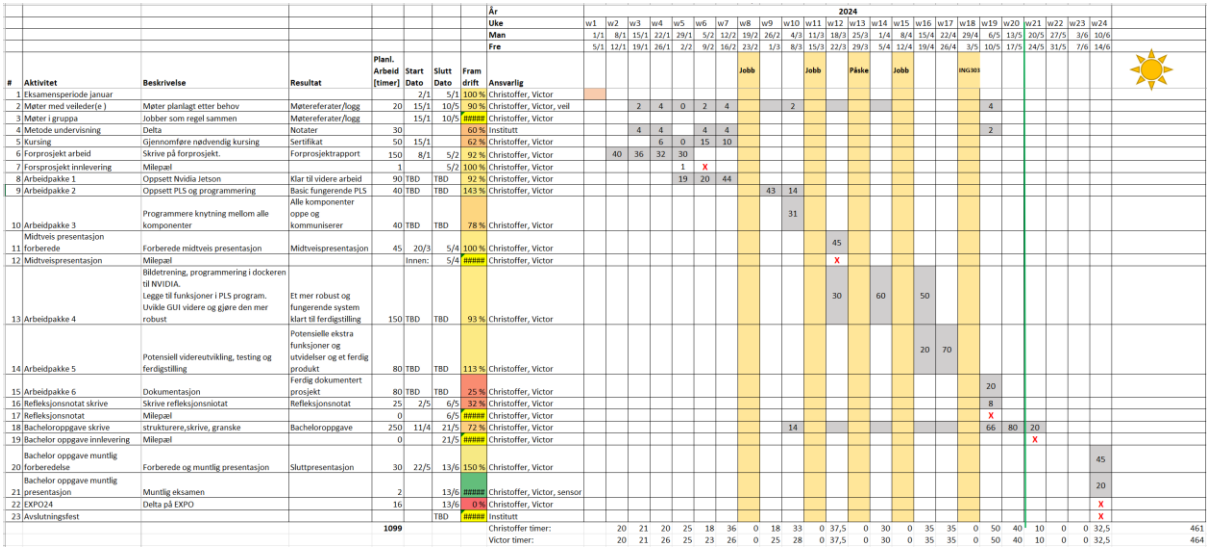
A.3 Fremdriftsplan



#	Aktivitet	Beskrivelse	Resultat	Planl. Arbeid [timer]	Start Dato	Slutt Dato	Fram drift	Ansvarlig
1	Eksamensperiode januar				2/1	5/1	100 %	Christoffer, Victor
2	Møter med veileder(e)	ukentlig? møte	møtereferater/logg	40	15/1	10/5	15 %	Christoffer, Victor, veil
3	Møter i gruppa	ukentlig? møte	møtereferater/logg	40	15/1	10/5	0 %	Christoffer, Victor
4	Metode undervisning	delta på	Notater	30			40 %	institutt
5	Kursing	Gjennomføre nødvendig kursing	Sertifikat	80	15/1		8 %	Christoffer, Victor
6	Forprosjekt arbeid	Skrive på forprosjekt. Diverse prosjektplanlegging.	Forprosjektrapport	150	8/1	5/2	95 %	Christoffer, Victor
7	Forsprosjekt innlevering	milepæl		0		5/2	#####	Christoffer, Victor
8	Arbeidpakke 1	Starte med Nvidia Jetson	TBD	80	TBD	TBD	25 %	Christoffer, Victor
9	Arbeidpakke 2	TBD	TBD	80	TBD	TBD	0 %	Christoffer, Victor
10	Arbeidpakke 3	TBD	TBD	80	TBD	TBD	0 %	Christoffer, Victor
11	Midtveis presentasjon forberede	Forberede midtveis presentasjon	Midtveispresentasjon	40	20/3	5/4	0 %	Christoffer, Victor
12	Midtveispresentasjon	Milepæl		0	Innen:	5/4	#####	Christoffer, Victor
13	Arbeidpakke 4	TBD	TBD	80	TBD	TBD	0 %	Christoffer, Victor
14	Arbeidpakke 5	TBD	TBD	80	TBD	TBD	0 %	Christoffer, Victor
15	Arbeidpakke 6	TBD	TBD	80	TBD	TBD	0 %	Christoffer, Victor
16	Refleksjonsnotat skrive	Skrive	Refleksjonsnotat	15	2/5	6/5	0 %	Christoffer, Victor
17	Refleksjonsnotat	Milepæl		0		6/5	#####	Christoffer, Victor
18	Bacheloroppgave skrive	strukturere,skrive, granske	Bacheloroppgave	150	11/4	21/5	0 %	Christoffer, Victor
19	Bachelor oppgave innlevering	Milepæl		0		21/5	#####	Christoffer, Victor
20	Bachelor oppgave muntlig forberedelse	Forberede og muntlig presentasjon	Sluttpresentasjon	30	22/5	13/6	0 %	Christoffer, Victor
21	Bachelor oppgave muntlig presentasjon	Muntlig eksamen		2		13/6	0 %	Christoffer, Victor, sensor
22	EXPO24	Delta på EXPO		16		13/6	0 %	Christoffer, Victor
23	Avslutningsfest					TBD	#####	institutt
				1073				Christoffer timer:
								Victor timer:

Figur 30 - Første utkast Gantt-diagram

Opprinnelig Gantt-diagram fra 4. uken ut i prosjekt se Figur 30.



#	Aktivitet	Beskrivelse	Resultat	Planl. Arbeid [timer]	Start Dato	Slutt Dato	Fram drift	Ansvarlig
1	Eksamensperiode januar				2/1	5/1	100 %	Christoffer, Victor
2	Møter med veileder(e)	Møter planlagt etter behov	Møtereferater/logg	20	15/1	10/5	90 %	Christoffer, Victor, veil
3	Møter i gruppa	Jobber som regel sammen	Møtereferater/logg		15/1	10/5	#####	Christoffer, Victor
4	Metode undervisning	Delta	Notater	30			60 %	institutt
5	Kursing	Gjennomføre nødvendig kursing	Sertifikat	50	15/1		62 %	Christoffer, Victor
6	Forprosjekt arbeid	Skrive på forprosjekt.	Forprosjektrapport	150	8/1	5/2	92 %	Christoffer, Victor
7	Forsprosjekt innlevering	Milepæl		1		5/2	100 %	Christoffer, Victor
8	Arbeidpakke 1	Oppsett Nvidia Jetson	Klar til videre arbeid	90	TBD	TBD	92 %	Christoffer, Victor
9	Arbeidpakke 2	Oppsett PLS og programmering	Basic fungerende PLS	40	TBD	TBD	143 %	Christoffer, Victor
10	Arbeidpakke 3	Programmere knytning mellom alle komponenter	Alle komponenter oppe og kommuniserer	40	TBD	TBD	78 %	Christoffer, Victor
11	Midtveis presentasjon forberede	Forberede midtveis presentasjon	Midtveispresentasjon	45	20/3	5/4	100 %	Christoffer, Victor
12	Midtveispresentasjon	Milepæl			Innen:	5/4	#####	Christoffer, Victor
13	Arbeidpakke 4	Bildetrening, programmering i dockeren til NVIDIA. Legge til funksjoner i PLS program. Uvikle GUI videre og gjøre den mer robust	Et mer robust og fungerende system klart til ferdigstilling	150	TBD	TBD	93 %	Christoffer, Victor
14	Arbeidpakke 5	Potensiell videreutvikling, testing og ferdigstilling	Potensielle ekstra funksjoner og utvidelser og et ferdig produkt	80	TBD	TBD	113 %	Christoffer, Victor
15	Arbeidpakke 6	Dokumentasjon	Ferdig dokumentert prosjekt	80	TBD	TBD	25 %	Christoffer, Victor
16	Refleksjonsnotat skrive	Skrive refleksjonsnotat	Refleksjonsnotat	25	2/5	6/5	32 %	Christoffer, Victor
17	Refleksjonsnotat	Milepæl		0		6/5	#####	Christoffer, Victor
18	Bacheloroppgave skrive	strukturere,skrive, granske	Bacheloroppgave	250	11/4	21/5	72 %	Christoffer, Victor
19	Bachelor oppgave innlevering	Milepæl		0		21/5	#####	Christoffer, Victor
20	Bachelor oppgave muntlig forberedelse	Forberede og muntlig presentasjon	Sluttpresentasjon	30	22/5	13/6	150 %	Christoffer, Victor
21	Bachelor oppgave muntlig presentasjon	Muntlig eksamen		2		13/6	#####	Christoffer, Victor, sensor
22	EXPO24	Delta på EXPO		16		13/6	0 %	Christoffer, Victor
23	Avslutningsfest					TBD	#####	institutt
				1099				Christoffer timer:
								Victor timer:

Figur 31 - Siste utkast Gantt-diagram

Endelig Gantt-diagram fra ferdig prosjekt se Figur 31.

For komplett Gantt-diagram og timelister se <Vedlegg 8>.

Appendiks B Brukerdokumentasjon

B.1 Brukerdokumentasjon

<Se vedlegg 1 og 7>

Appendiks C Bill Of Materials

Beskrivelse:	Pris:	Betales av
NVIDIA Jetson Orin Nano	7580.31, -	Bedrift
HAMA MICROSDXC 128 GB C10 UHS-I 80 MB/S	199,-	Studenter
3-printet brakett DIN NVIDIA Jetson	10,-	Studenter
CodeAcademy Python kurs	700,-	Studenter
Diverse materiell til oppkobling	0,-	Lånes etter behov av bedrift
CVAT 2måneder abonnement	1000,-	Studenter