Nr. _____ av _____

Sjøkrigsskolen

Bacheloroppgave

Implementering av objektdeteksjon på maritim dronesverm

– Bruk av kunstig intelligens til maritim overvåkning på ubemannede fartøy –

av

Olav Hollup og Magnus Røsand

Levert som en del av kravet til graden:

BACHELOR I MILITÆRE STUDIER MED FORDYPNING I LEDELSE OG MARINEINGENIØR VÅPEN, ELEKTRONIKK OG DATA

Innlevert: Desember 2022

Godkjent for offentlig publisering

Publiseringsavtale

En avtale om elektronisk publisering av bachelor/prosjektoppgave

Kadettene har opphavsrett til oppgaven, inkludert rettighetene til å publisere den.

Alle oppgaver som oppfyller kravene til publisering vil bli registrert og publisert i Bibsys Brage når kadettene har godkjent publisering.

Oppgaver som er graderte eller begrenset av en inngått avtale vil ikke bli publisert.

Vi gir herved Sjøkrigsskolen rett til å gjøre denne oppgaven tilgjengelig elektronisk, gratis og uten kostnader	[X] Ja	[ ] Nei
Finnes det en avtale om forsinket eller kun intern publisering? (Utfyllende opplysninger må fylles ut)	[ ] Ja	[X] Nei
Hvis ja: kan oppgaven publiseres elektronisk når embargoperioden utløper?	[ ] Ja	[ ] Nei

Plagiaterklæring

Vi erklærer herved at oppgaven er mitt eget arbeid og med bruk av riktig kildehenvisning.

Vi har ikke nyttet annen hjelp enn det som er beskrevet i oppgaven.

Vi er klar over at brudd på dette vil føre til avvisning av oppgaven.

Dato: 18 – 12 – 2022

Magnus Svensson Røsand
Kadett navn

Kadett, signatur

Olav Hollup
Kadett navn

Kadett, signatur

Forord

Bacheloroppgaven er skrevet av Olav Hollup og Magnus Svensson Røsand i perioden august 2022 til desember 2022. Den er skrevet som en avsluttende del av vår bachelorgrad ved FHS Sjøkrigsskolen.

Oppgaven omhandler implementering av objektdeteksjon på dronesverm for å kunne drive maritim overvåkning. Prosjektet har satt krav til å sette seg inn i store mengder ukjent eller tilnærmet ukjent teori for å svare til målene som ble satt. Maskinlæring er noe vi kun har hatt et introduksjonsemne om på skolen og var dermed krevende å skjønne seg på. Det var også krevende å sette seg inn i en tidligere bacheloroppgave. Likevel har det vært interessant å lære seg noe helt nytt og å gjøre et dypdykk i avansert teknologi. Oppgaven har gitt mersmak for ytterligere arbeid med programmering og åpnet øynene for hvor tilgjengelig informasjonen er på nettet.

Leseren antas å ha kjennskap til grunnleggende programmering i Python og en grunnleggende kunnskap innenfor maskinlæring.

I denne oppgaven har vi mange å takke. Først ønsker vi å gi en ekstra stor takk til førsteamanuensis Alexander Sauter for veiledning, konstruktive samtaler og hjelp underveis i prosjektet. Vi ønsker også å takke førsteamanuensis Christophe Massacand for fruktbare diskusjoner rundt maskinlæring og andre oppdukkende temaer. En takk må også rettes til førsteamanuensis Linn-Kristine Glesnes Gaupholm for gode og konstruktive tilbakemeldinger.

Bergen, Sjøkrigsskolen, 19-12-2022

Olav Hollup

_____

Magnus Svensson Røsand

_____

Sammendrag

Norge har verdens nest lengste kyst og det er dermed utfordrende å opprettholde situasjonsforståelse langs kysten til enhver tid. Samtidig er det signalisert i Langtidsplanen for Forsvaret at antallet bemannede fartøy vil reduseres (Hareide, Relling, Pettersen, Sauter, & Mjelde, 2018). For å bedre situasjonsforståelsen kan ubemannede overflatefartøy bli benyttet. Likevel vil enkeltenheter bare kunne dekke enkeltområder av gangen. Et godt alternativ er da sverm. Tre eller flere enheter som jobber sammen for å gjennomføre arbeidsoppgaver. I 2019 ble det utviklet en testplattform for dronesverm (Hellesnes & Lyssand, 2019). Konseptet kan føre til dekning i store områder om enhetene jobber sammen for å løse en oppgave. Sammenhengen mellom et behov for situasjonsforståelse langs kysten og svermoppgaven fra 2019 fører oss inn på vår problemstilling:

Hvordan kan objektgjenkjenning implementeres på dronesvermer for å gjøre deteksjoner og identifikasjoner av fartøy?

Prosjektet har utviklet et konsept for maritim overvåkning på dronesverm gjennom bruk av objektdeteksjon. For å bevise konseptet har vi videretrent en maskinlæringsmodell til å gjenkjenne en egendefinert fartøysklasse. Resultatene fra testen har vært lovende og vist at avansert teknologi kan anvendes med begrensede ressurser innenfor objektdeteksjonsfeltet. Oppgaven har også belyst hvor avhengig objektdeteksjonsmodeller er på endringer av miljø. Det anbefales derfor å starte å bygge opp datasett med bilder av norskekysten og på fartøy av interesse, slik at Sjøforsvaret er klar for implementering av lignende teknologi i fremtiden.

Videre benytter prosjektet data fra maskinlæringsmodellen til å regne ut relativ peiling. Dette gjør det mulig for enhetene i svermen å kunne lokalisere detekterte fartøy. Denne informasjonen blir deretter delt med de andre svermenhetene. Det kan derfor tenkes at sammensetninger av relative peilinger, kan benyttes for å finne absolutt posisjon på et detektert fartøy. En svermalgoritme som legger til rette for dette er enda ikke laget og anbefales som videre arbeid fra dette prosjektet.

Innholdsfortegnelse

Figurer

Tabeller

Nomenklatur

Begrep/forkortelse	Beskrivelse
CNN	Convolutional neural network
ML	Machine Learning
mAP	Mean average precision
Gradient descent	Gradientnedstigning
Input	Inngang
TF	TensorFlow maskinlæringsplatform
TFLite	TensorFlow Lite maskinlæringsplatform
NAS	Neural architecture search
Array	Samling av elementer som hører til en spesifikk kategori
GCS	Ground Control Station
RPi	Raspberry Pi
FPS	Frames per second

1 Innledning

Dette kapittelet vil presentere oppgavens rammer som innebefatter bakgrunn, problemstilling, mål, begrensninger og struktur.

1.1 Bakgrunn

Norge har verdens nest lengste kyst og et lite sjøforsvar til å verne om det. Kystområdet innbefatter 100 915 kilometer og i dag «er Norge en ledende hav- og kystnasjon.» (Helleland & Astrup) For å verne kystlinjen er Sjøforsvaret essensielt. Dagens styrkestruktur består av større bemannede fartøy, men likevel er det signalisert i Langtidsplanen for Forsvaret at antallet bemannede fartøy vil reduseres (Hareide, Relling, Pettersen, Sauter, & Mjelde, 2018). Med bakgrunn i dette kan det være plausibelt å anta at dagens styrkestruktur ikke gir tilstrekkelig situasjonsforståelse langs kysten til enhver tid.

Ubemannede systemer og automatisering kan være et godt tilskudd dersom de anvendes i en operativ kontekst. Implementering av autonome plattformer kan bidra til å øke situasjonsforståelsen og gi et bedre grunnlag for å verne kystområdet (Hareide, Relling, Pettersen, Sauter, & Mjelde, 2018). Et mulig tilskudd til flåtestrukturen er ubemannede overflatefartøy som driver maritim overvåkning. Et slikt tilslag til flåtestrukturen vil opprette og vedlikeholde oversikt i begrensede geografiske områder. Ved utplassering i flere områder vil dette potensielt føre til et bedret oversiktsbilde langs kysten.

Likevel kan det argumenteres for at ubemannede fartøy som opererer alene ikke klarer å dekke store områder effektivt og er enkelt for en fiende å håndtere. En teknologi som tilføyer disse dimensjonene til alternativet, er sverm. Sverm er sammensetninger av tre eller flere enheter som kan operere sammen med minimal ekstern påvirkning (Hellesnes & Lyssand, 2019). Hver enhet tar beslutninger med hensyn på data fra miljøet den opererer i, ut ifra satte regler for enheten. At en svermer er sammensetninger av flere enheter gjør dem vanskeligere å håndtere for en fiende. Videre fører flere enheter, som arbeider sammen, til at svermen som helhet kan dekke større områder effektivt. Det er også viktig å poengtere at dette konseptet baserer seg på enkle sjødroner som virker sammen for å oppnå et felles mål. Følgelig fører dette til at plattformene er billige. I 2019 ble det utviklet en testplattform for dronesverm på FHS Sjøkrigsskolen av Andreas Hellesnes og Kim Lyssand (Hellesnes & Lyssand, 2019). Dronesvermen tar utgangspunkt i prinsippene

over, og er et godt utgangspunkt for å utvikle et konsept for maritim overvåkning på ubemannede systemer.

For å gjennomføre overvåkning av området kan kunstig intelligens, mer spesifikt objektdeteksjon, integreres på plattformene. Teknologien bak er i sterk utvikling og benyttes i stadig større grad i flere samfunnssektorer. I sivil sektor legger store bedrifter, som Google, ressurser inn i videreutvikling av teknologien. Dette har ført til en oppgang i åpen kildekode for utviklere. Enkelte av disse krever at brukeren har relativt lav bakgrunnskunnskap og kan dermed integreres enkelt med gode resultater.

Det kan derfor tenkes at dronesvermer med objektdeteksjon kan overvåke norske fjorder. Et system som kan detektere, lokalisere og videresende informasjon for å gjøre situasjonsforståelsen langs kysten bedre.

Med utgangspunkt i ny teknologi og testplattformen fra 2019, fører dette oss inn på vår problemstilling**:**

Hvordan kan objektgjenkjenning implementeres på dronesvermer for å gjøre deteksjoner og identifikasjoner av fartøy?

Oppgaven presenterer et konsept for maritim overvåkning ved hjelp av dronesvermer. Videre i oppgaven tar vi for oss mål for konseptet med tilhørende begrensninger samt strukturen. En viktig bemerkning er at dronesvermen ikke vil være i hovedfokus gjennom oppgaven. Hovedfokuset vil være implementering av et system for deteksjon og identifikasjon av fartøy på dronesverm.

1.2 Mål

Objektdeteksjon og dens bruksområder er et stort fagområde og av den grunn må målene for prosjektet vårt konkretiseres. Disse målene er summert opp i tabell 1. Intensjonen med prosjektet er å fremheve hvor enkelt slik teknologi kan implementeres med dagens tilgjengelige åpen kildekode.

Mål	Beskrivelse
Deteksjon/Identifikasjon	Kunne detektere egendefinert klasse med presisjon høyere eller lik 80% mAP
Lokalisering	Enhetene skal internt kunne regne ut relativ peiling ved hjelp av kamera og deteksjonsmodellen
Deling	Deteksjonsdata skal kunne kommuniseres til andre enheter i svermen samt operatørpanel.
Samler deteksjonsdata til videre trening	Enhetene skal kunne lagre bilder ved deteksjon for videre trening av deteksjonsalgoritme.

Tabell 1: Mål for oppgaven

1.3 Begrensninger

En begrensning i oppgaven er at den fokuserer på én gruppe av objektdeteksjonsmodeller som en løsning på problemet. Det er ikke valget av maskinlæringsmodell som er essensielt i oppgaven.

Videre begrenses oppgaven ved at den kun tar utgangspunkt i åpen kilde-alternativer. Med dette gjelder data- samt kildekoder. Med data menes bildene modellen benytter som datasett.

Trening av maskinlæringsmodeller er en tidkrevende prosess og vil dermed også være en avgrensning i oppgaven. Oppgaven vil ta i bruk åpen kilde-modeller og drive videretrening av disse modellene.

Måten objekter detekteres på er også en begrensning. I denne oppgaven vil det kun bli presentert bruk av kamera som deteksjonssensor. Andre sensorer, som termiske kameraer, kunne også bli benyttet.

Vår bruk av objektdeteksjon er også avgrenset. Objektedeteksjon kunne også blitt benyttet til kollisjonsunnvikelse.

Videre inn på dette vil oppgaven også begrenses til implementeringen av objektdeteksjon på testplattformen fra 2019. Deteksjonsdataen vil kun inngå i svermens kommunikasjonsgang, men den vil her ikke agere på deteksjonsdata.

Datamaskinene som kjører maskinlæringsmodellen, er også en begrensning i form av tilgjengelig prosessorkraft. Vi har benyttet oss av Raspberry Pi for å bevise konseptet.

Oppgaven vil kun ta for seg bildebygging i overflatedomenet som bruksområde for konseptet. Det vil derfor ikke bli diskutert hvorvidt konseptet fungerer i flere krigføringsdomener.

Oppgaven søker å gi et konseptuelt resultat for maritim overvåkning i mindre skala. Dette gjør at oppgaven ikke presenterer en ferdig løsning, men et konsept som kan være til nytte for videre utvikling inn mot Sjøforsvarets behov.

1.4 Struktur

Oppgaven er delt inn i seks kapitler. Innledningen forklarer bakgrunnen for valg av oppgave, definerer oppgavens problemstilling og setter rammene rundt oppgaven. Deretter forklares teorien som er nødvendig for å forstå den tekniske løsningen som er valgt. Første del av teorien presenterer essensiell programvare og generelle begreper innen maskinlæring. Dette gir grunnlaget for å forstå objektdeteksjon ved hjelp av maskinlæring.

I kapittel tre forklares implementeringen av den tekniske løsningen. Videre tester og presenterer vi resultatene på den tekniske løsningen. Resultatene er delt opp i tre deler. Del en inneholder resultater fra testing av maskinlærings-modeller. Den andre delen beskriver testene av enkelte systemdeler som kommunikasjon og utregning av relativ peiling på deteksjoner samt operatørpanelet. Den tredje delen handler om lagring av data for videretrening av maskinlærings-modeller. I kapittel fem blir den tekniske løsningen drøftet opp mot resultater, samt ~~potensialet~~ konseptet kan ha for maritim overvåking. Det siste kapitelet konkluderer oppgaven og gir videre anbefalinger for ytterligere arbeid.

2 Teori

Dette kapitelet vil trekke frem nødvendig teori for å forstå hvordan konseptet for maritim overvåkning har blitt implementert. Kapittelet starter derfor med å presentere svermoppgaven fra 2019. (Hellesnes & Lyssand, 2019) Dette skal gi en forståelse for hvordan svermen fungerer, og derav større forståelse for plattformen vi har implementert objektdeteksjon på. Videre vil metaoperativsystemet svermoppgaven er bygget på bli presentert. Resten av kapittelet vil fokusere på grunnleggende prinsipper innen maskinlæring samt ulike typer maskinlæring for å kunne forstå objektdeteksjon.

2.1 Svermoppgaven fra 2019

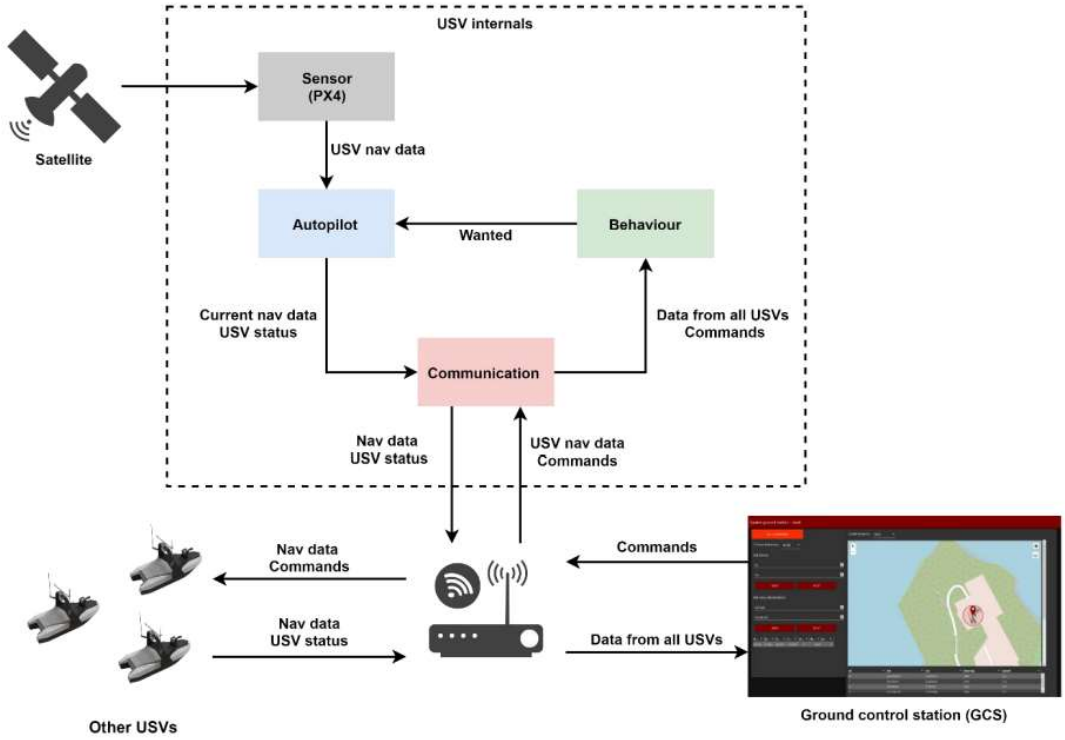
«Begrepet sverm kommer fra naturen som en beskrivelse av hvordan dyr av forskjellige arter danner grupperinger for å overleve og løse oppgaver de ikke klarer alene.» (Hellesnes & Lyssand, 2019)

I 2019 ble det utviklet en testplattform for dronesverm (Hellesnes & Lyssand, 2019). Plattformen inkluderer en fungerende svermalgoritme og er av den grunn et godt grunnlag for å utvikle plattformen ytterligere. Hellesnes og Lyssand (2019) anbefalte å tilføre økt miljøoppfatning i svermen. Det er denne delen vi ønsker å bygge videre på, men for å forstå svermens funksjon er det noen begreper som bør defineres og bli gjort rede for.

Plattformen er basert på sjødroner utviklet ved FHS SKSK. På disse enhetene finnes det en Raspberry Pi som sammen med en Arduino styrer 3 motorer og 2 servoer for 3. Tilkoblet Raspberry Pi er en Pixhawk som sensorsystem med blant annet GPS modul, kompass og akselerometer. Pixhawk benyttes til å finne egen posisjon samt bevegelsesdata. Egen posisjon og bevegelsesdataen benyttes internt for styring av båten og kommuniseres ut til de andre enhetene i svermen.

For å gjøre dronene til en sverm tar hver enhet beslutninger basert på egen og andres GPS-informasjon. Kommunikasjonen er derfor helt essensiell for at de skal kunne bevege seg i samspill med hverandre. Videre er dette støttet av en autopilot som sørger for styringen av hver enkelt enhet og en beslutningstager som bestemmer hvilken atferd svermen skal operere i. Atferden er et begrep som forklarer hvordan enhetene skal operere i forhold til hverandre.

Kommunikasjonen mellom enhetene og måten dette styrer systemet er vist i figur 2-1. PX4 er GPS-modulen. Denne henter inn posisjons- og bevegelsesdata for den enkelt enhet. Autopiloten bruker data fra denne samt hvilken ønsket posisjon den skal oppnå, ifølge adferdslogikken som kalles her «Behaviour», til å finne ny styringsdata. Ny styringsdata sendes videre som del av kommunikasjonen, som videreformidles til resten av svermen via delt WiFi. Kommunikasjonen på hver enhet henter også posisjons- og bevegelsesdata fra de andre enhetene i svermen samt «Commands». «Commands» styres fra «Ground control station» og bestemmer blant annet hvilken atferd som skal benyttes. Fra kommunikasjonen videreformidles andre enheters GPS-data og hvilken atferd som er bestemt til «Behaviors». Med bakgrunn i disse dataene bestemmer «Behaviors» hva som er ønsket styring for hver sjødrone.



Figur 2-1: Systemstruktur av systemer i svermen. Bildet er hentet fra (Hellesnes & Lyssand, 2019).

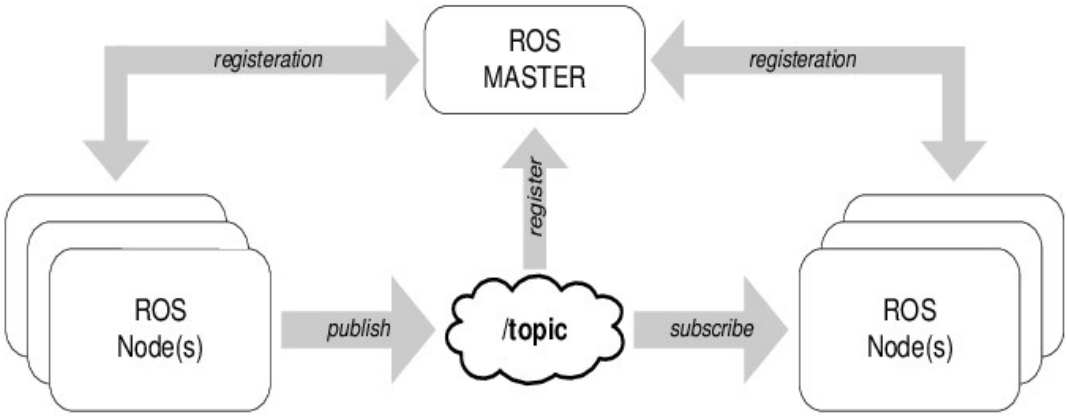
2.2 Robot Operating System

Svermoppgaven fra 2019 benytter Robot Operaing System for å kjøre koden til dronesvermen. Robot Operating System, forkortet ROS, er et metaoperativsystem laget for roboter. Programvaren innebefatter blant annet verktøy og biblioteker for å skaffe, bygge, skrive og kjøre kode fordelt over flere datamaskiner (Open Robotics, 2018). Dette gjør ROS til en egnet plattform for kommunikasjon mellom flere enheter. Videre i dette delkapittelet presenterer vi nødvendige deler av ROS for å gi grunnlag for å forstå hvordan deteksjonsdata kan kommuniseres inn i svermkommunikasjonen.

ROS består blant annet av en *master*, flere program-noder, *messages* og *topic's*. Programnodene får en adresse fra *master* for å kommunisere med de andre nodene. *Master* har full oversikt over adressene og kan dermed koble nodene sammen. (Open Robotics, 2018) En node er et program som styrer en del av systemet. Dette kan eksempelvis være en node som styrer motorsystemet og en node som sørger for styringen.

ROS er laget for bruk på roboter som gjerne kjører flere moduler samtidig. En modul er programmer som kjøres i systemet. Hver modul styres av en node. Sammensetningen av moduler gjør at systemet fungerer helhethetlig. (Fairchild & Harman, 2016) Disse prosessene kan være svært sensortunge for å forstå miljøet systemet operer i. Uten kommunikasjon mellom disse prosessene ville dermed systemet ikke samhandler enhetlig. Et eksempel på dette er kjøring av autonome biler uten kommunikasjon med sensormoduler som analyserer miljøet rundt. Bilen hadde da kjørt blindt rundt og det er derfor nødvendig med et samspill mellom modulene.

Måten nodene kommuniserer på, er ved å *publisere* og *abonnere* på informasjon tilordnet forskjellige *topics*. Dette er vist i figur 2-2. Kun én node kan *publisere* til et *topic*, men flere noder kan *abonnere* på samme *topic* (Fairchild & Harman, 2016). Ved å *publisere* sendes det data til en gitt adresse og fra denne adressen kan andre noder lese av data som er *publisert*. På denne måten kommuniserer nodene og sørger for at de ulike delprosessene har informasjonen de trenger.



Figur 2-2: Visualisering av ROS kommunikasjon. Hentet fra (Achmad, Priyandoko, Roali, & Daud, 2017)

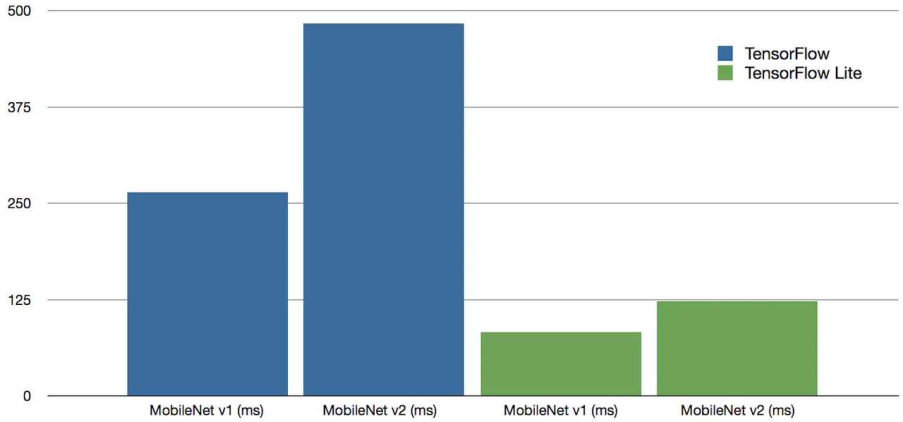
Messages definerer datatypene til variablene som skal *publiseres* til *topics*. *Messages* er en strukturell oppbygning av informasjonen som skal *publiseres.* (Open Robotics, 2018) Strukturen i én *message* inneholder forskjellig informasjon i underkategorier som *topic* skal inneholde. Disse underkategoriene strukturerer datatypene opp mot tilhørende variabler i én *message* for at de andre nodene skal kunne tolke informasjonen.

2.3 Tensorflow Lite

Tensorflow er en end-to-end åpenkilde-plattform for maskinlæring (Abadi, et al., 2015). Plattformen tilbyr et omfattende og fleksibelt økosystem av biblioteker, verktøy og ressurser laget av Googles utviklere. Plattformen gjør det enkelt å bruke ferdig utviklede maskinlæringsmodeller, videretrene eksisterende modeller eller lage nye modeller. Tensorflow støtter ulike programmeringsspråk, som gjør den fleksibel og den kan anvendes på forskjellige bruksområder innen maskinlæring. I denne oppgaven benytter vi åpen kilde-kode fra Tensorflow og programmeringsspråket Python vil hovedsakelig bli benyttet.

Tensorflow Lite er utgaven av Tensorflow som er tilpasset bruk på mobiltelefoner og andre såkalte *edge devices*. Et eksempel på et *edge device* er Raspberry Pi. Svermenhetene i denne oppgaven har begrenset datakraft og lokal lagringsplass. Det er derfor viktig at vi ender opp med en maskinlæringsmodell som er lite krevende og samtidig effektiv. Tensorflow Lite plattformen er en lettvekts utgave av Tensorflow. Den har et mindre utvalg av verktøy å sette i bruk, men den kjører modeller raskere enn vanlig TensorFlow.

(Abadi, et al., 2015) I figur 2-3 er det lagt frem en sammenligning mellom Tensorflow og Tensorflow Lite målt i millisekunder under kjøring av to objektdeteksjonsmodeller på RPi 4. Her kommer det frem at Tensorflow Lite kan kjøre deteksjonsmodellen raskere enn Tensorflow. Kostnaden man betaler for dette er lavere prediksjonsnøyaktighet.

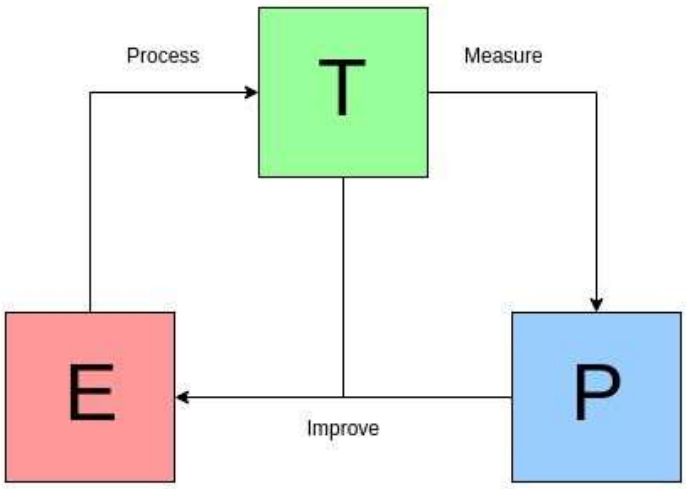


Figur 2-3: Sammenligning av Tensorflow og Tensorflow Lite på Raspberry Pi 4 (Allan, 2019)

2.4 Machine learning

I 1959 definerte Arthur Samuel maskinlæring som et "field of study that gives computers the ability to learn without being explicitly programmed" (Ng, 2012). Dette er en eldre og mindre formell definisjon på feltet, men er likevel visende. Maskinlæring er et felt der man gir maskinen forutsetningene den trenger, for å selv kunne løse en satt oppgave.

En annen, mer presis og moderne definisjon ble formidlet av Tom Mitchell. Han forklarte maskinlæring som følgende. «A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E." (Ng, 2012). En visualisering av dette vises i figur 2-4.



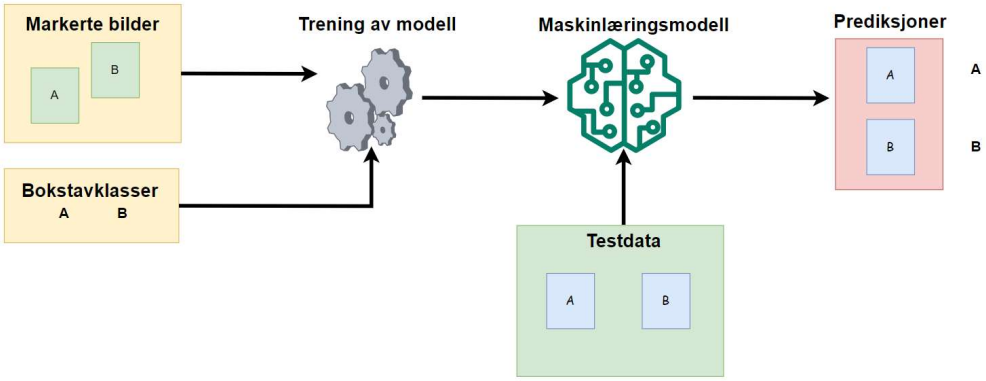
Figur 2-4: Visualiering av maskinlæring (Mayo, 2018)

Definisjonen kan forklares med et sjakkprogram. Erfaringen E er det programmet har lært etter å ha spilt et visst antall spill med sjakk. Arbeidsoppgaven T er å spille sjakk, og ytelsesmålingen P er hvor god maskinen er til å spille sjakk. Maskinen kan derfor sies å lære av sin erfaring med å spille sjakk dersom maskinen blir bedre etter hvert som den spiller flere partier.

En maskin bygger opp erfaringen sin ved hjelp av:

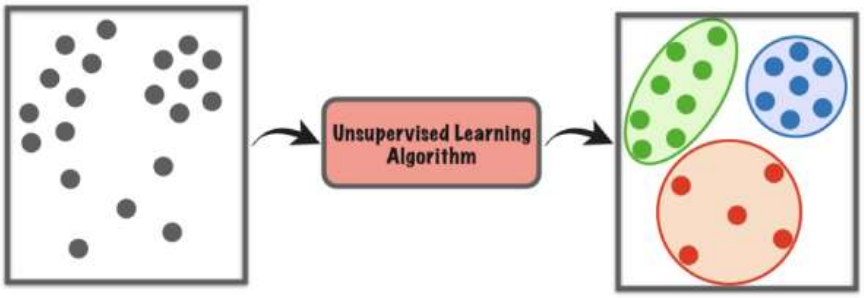
1. Veiledet maskinlæring eller …
2. Uveiledet maskinlæring

Ved veiledet maskinlæring veiledes maskinen til å forstå sammenhengen mellom input og output. Dette gjøres ved at en person presenterer rett løsning for maskinen under trening (IBM Cloud Education, 2020). Konseptet kan forklares med et eksempel. Maskinen skal gjenkjenne bokstaver i et bilde. Trenings-datasettet vil da være bilder som inneholder bokstaver. Veileder markerer hvor bokstavene er i bildet og beskriver hvilken bokstav dette er. Dette gjøres under trening og maskinen vil da lære seg sammenhengen mellom input og output. På denne måten vil maskinen kunne estimere hvilken bokstav som er A og B etter trening. Dette vises i figur 2-5.



Figur 2-5: Veiledet maskinlæring med prediksjoner

Uveiledet maskinlæring fungerer uten en veiledende part. En slik algoritme kan utvikles til å forstå seg på sammenhenger i data, som operatøren av systemet ikke nødvendigvis vet resultatet på selv (Ng, 2012). Modellen blir utviklet ved at algoritmen samler bolker med data der den ser sammenheng mellom forskjellige inputverdier. Et eksempel på bruksområde for dette er i sammenligning av gener. Ved uveiledet maskinlæring vil genene bli gruppert i bolker der sammenhengen mellom variabler er relativt like. Eksempler på slike variabler er levetid, lokasjon og roller. Figur 2-6 viser hvordan sammenhenger i data kan settes sammen i bolker.

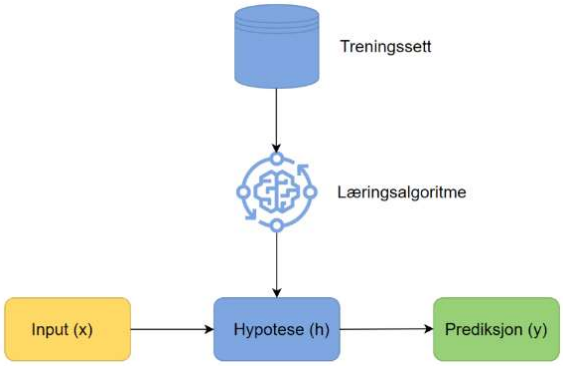


Figur 2-6: Uveiledet maskinlæring. Hentet fra (Jeffares, 2018).

Videre i denne oppgaven vil veiledet maskinlæring ha fokus ettersom at det blir et sentralt verktøy for resten av oppgaven. Av den grunn vil vi gå dypere inn på veiledet maskinlæring videre i teoridelen.

2.4.1 Modellrepresentasjon

I dette delkapittelet skal vi ta for oss modellrepresentasjon for veiledet maskinlæring. Dette vil bli forklart gjennom figur 2-7. Første delen i modellen er treningssettet. Treningssettet er et datasett som inneholder en input (x) og en output (y) per treningseksempel. I treningssettet finnes det et visst antall treningseksempler. Læringsalgoritmen bruker alle disse til å trene opp en hypotese. Hva som skjer inni denne læringsalgoritmen, vil vi komme tilbake til.



Figur 2-7: Visulisering av modellrepresentasjon. Inspirasjon hentet fra (Ng, 2012)

Hypotesen (h) er et matematisk uttrykk for sammenhengen mellom input og output. Maskinen kan dermed predikere output fra en input gjennom hypotesen. Således blir målet for veiledet maskinlæring å trene hypotesefunksjonen til å gjøre gode prediksjoner på output til en gitt input (Ng, 2012). Hypotesefunksjonen for regresjon er vist under:

$$h_\theta(x) = \theta_0 + \theta_1 x$$

2.4.2 Regresjon og klassifikasjon

Maskinlæring er til for å løse ulike problemer. To problemer maskinlæring blir benyttet til er regresjon og klassifikasjon. Regresjonsproblemer er problemer med reelle eller kontinuerlige verdier som output. Slik som andre maskinlæringsproblemer, har det en input som prediksjonene gjøres ut ifra (Shukla, 2022). Et eksempel på et regresjonsproblem er leilighetspriser. Her kan *features* være antall rom og totalt bruksareal, mens output vil være en prisprediksjon på boligen (Ng, 2012).

Klassifiseringsproblemer er problemer som har diskrete verdier som svar. Dette kan brukes til å sette svarene i forskjellige båser kalt klasser, eksempelvis ja eller nei, hund eller katt. I maskinlæring brukes klassifikasjon til blant annet mønstergjenkjenning. Slike algoritmer må ha mulighet til å lære og de må kunne generalisere (Egil, 2001). De kan lære ved at objekter som ligner på hverandre blir plassert i samme klasse, enten veiledet eller uveiledet. Dette fører til at en slik algoritme kan klassifisere usette objekter. (Egil, 2001)

2.4.3 Kostfunksjon

En kostfunksjon brukes til å vurdere om en hypotesefunksjon gir gode prediksjoner. Kostfunksjonen finner gjennomsnittlig differens mellom alle resultatene fra hypotesefunksjonen og hva resultatet i realiteten skulle ha vært. For å gjøre den bedre til å predikere bør differensen mellom output og reelt resultat minimeres (Ng, 2012). Mindre kostfunksjon gir mindre feil mellom estimert verdi og reell verdi. Derfor ønsker man en lav verdi på output av en kostfunksjon. Formel 2-1 viser en kostfunksjon for regresjonsproblemer. Variabelen *m* tilsvarer antall output fra modellen. Predikert verdi *h* blir subtrahert fra reell verdi *y* og finner differensen mellom dem.

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}\left(h\left(x^{(i)}\right) - y^{(i)}\right)^2$$

Formel 2-1: Kostfunksjon for regresjon

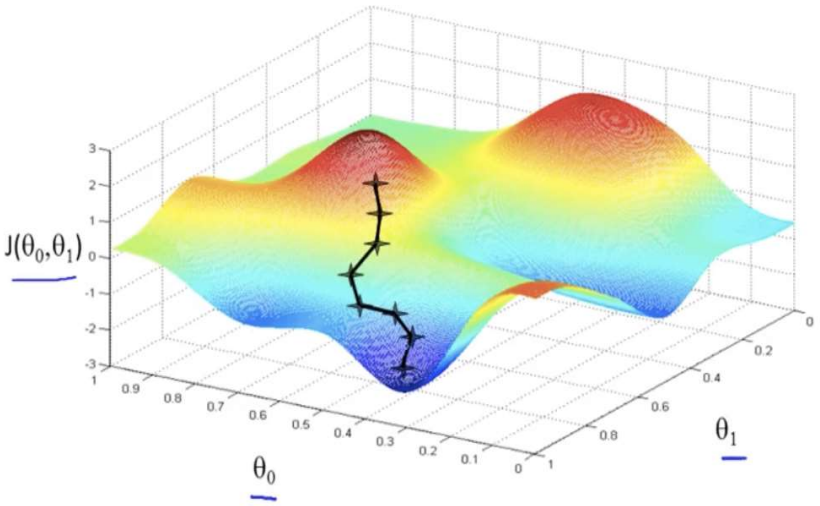
2.4.4 Gradient descent

Oppgaven har til nå gjennomgått hvordan man finner hypotesefunksjonen og metoden for å regne ut hvorvidt hypotesen er et godt predikat. For å minimere kostfunksjonen fører dette oppgaven inn i estimering av modellens parametere, θ. Parameterne er konfigurerte, interne variabler i modellen og kan estimeres fra gitte data (Paul, 2018). For å predikere et resultat fra en input er modellen avhengig av disse og fører oppgaven inn i gradient descent. For å forklare konseptet gjengis formelen for *gradient descent* med to kun en parameter.

$$\theta = \theta - \alpha\frac{\partial}{\partial\theta}J(\theta)$$

Gradient descent er ofte brukt innen maskinlæring for å minimere kostfunksjonen (Kwiatkowski, 2021). Dette gjøres ved å ta den deriverte av kostfunksjonen, J, for å finne hvilken retning våre parametere skal bevege seg mot. Stegene vil søke å gå mot den bratteste nedstigningen. Størrelsen på stegene bestemmes av læringsraten, α. For å finne ny parameter tar man original parameter og trekker fra ønsket endring for hvert steg. Disse stegene gjentas til konvergens, altså et lokalt minimum. Det er på dette tidspunktet parameterne er rett innstilt for oppgaven maskinlæringsmodellen skal løse. Figur 2-8viser et eksempel på *gradien descent*. Parameterne er plassert på horisontale akser og på vertikal

akse er kostfunksjonen. *Gradient descent* finner de optimale parameterne for en minst mulig kostfunkjson.



Figur 2-8: Visualisering av gradient decent. Bilde hentet fra (Ng, 2012)

2.4.5 Måleenheter for objektdeteksjon

Mean Average Precision (mAP) er et mål på hvor presis en maskinlæringsmodell er. mAP måles i prosent og tallet sier noe om hvor mange riktige prediksjoner modellen gjør i forhold til det totale antallet prediksjoner som er gjort (Liu, 2018). Dette vises i formel 2-2. I formelen brukes TP og FP, disse forkortelsene betyr sann positiv og falsk positiv. En sann positiv prediksjon samsvarer det reelle resultatet. En falsk positiv prediksjon er en utført prediksjon som ikke samsvarer med det reelle resultatet (Liu, 2018). mAR er altså er måleenhet man bruker om det er viktig å ikke gjøre falske deteksjoner.

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

Formel 2-2: Utregning av mAP

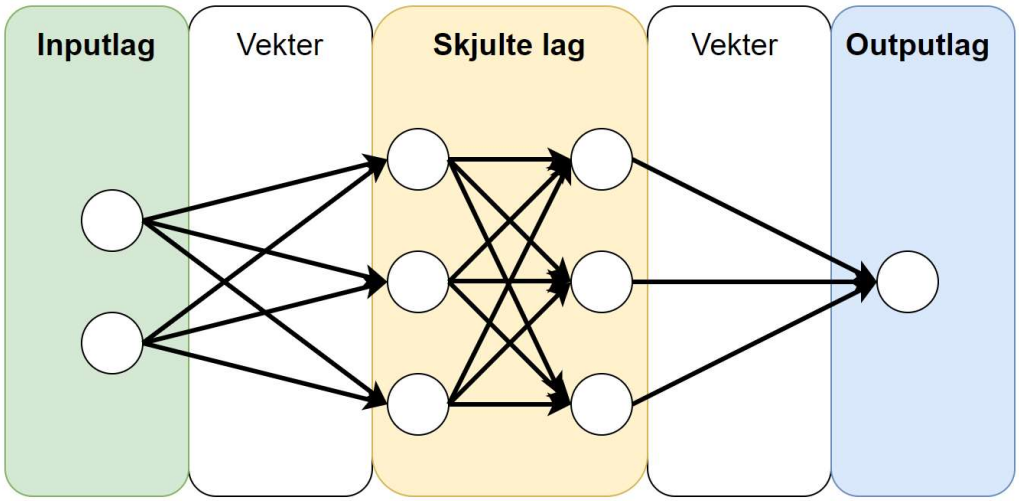
Average Recall (AR) er et annet mål på hvor presis en maskinlæringsmodell er. Men denne måleenheten ser på hvor mange riktige prediksjoner modellen gjør i forhold til alle riktige utfall. Riktige utfall vil si alle objekter i testdatasettet som skulle blitt detektert. Utregningen av AR er vist i Formel 2-3. TP betyr sann positiv prediksjon, og FN betyr falsk negativ prediksjon. Falsk negativ prediksjon er et objekt som ikke ble detektert. AR er altså en måleenhet man benytter om det er viktig å detektere alle objekter.

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

Formel 2-3: Utregning av AR

2.4.6 Neurale nettverk

Neurale nettverk er sammensetninger av algoritmer som søker å finne sammenhenger i datasett. Denne prosessen er inspirert av hvordan biologiske nervesystemer fungerer (Chen J. , 2022). Neurale nettverk inneholder sammensetninger av neuroner. Overgangen mellom neuronene skal sammensatt finne rett tolkning av inputdata for å kunne predikere reell output. Neuronene er delt inn i ett inputlag, ett eller flere skjulte lag, samt ett output-lag. Hvert lag kan inneholde flere noder. Nodene er sammenkoblet med samtlige noder i det neste laget, slik som illustrert i figur 2-9.



Figur 2-9: Visualisering av et nevralt nettverk

I hver enkelt kobling mellom noder er det en vekting (Ng, 2012). Vekting er en verdi utviklet av systemet ved trening og har samme funksjon som parameterne i 2.4.4. I et trent system forandres vektingen med hensyn på input og endrer verdien på hver enkelt node. Avhengig av input og vektingen vil nodenes verdi tilpasses til situasjonen. Dersom en eller flere noder overstiger en viss verdi vil noden aktiveres og sende signaler videre til nodene i neste lag (IBM Cloud Education, 2020). På denne måten bryter nettverket ned store mengder med data i de ulike lagene og finner en sammenheng mellom dem.

Størrelsen på nettverket er essensielt for bruksområdet. Mindre nettverk har færre overganger med tilhørende vektinger og fører dermed til at nettverket er mindre krevende å kjøre. Naturlig nok fører dette også til lavere presisjon i output. På den andre siden har større nettverk flere overganger, som fører til at nettverket er mer krevende å kjøre. Dette gir en mer nøyaktig presisjon på output.

2.4.7 Backward og forward propagation

Backward propagation er en viktig del for å lære opp et neuralt nettverk. Under trening kjøres input gjennom nettverket og det regnes ut en feil i output ved hjelp av en kostfunksjon. Etter feil i output er funnet søker konseptet å finne hvor mye hver enkelt vekting bidrar til output-feilen. Feilsøkingen starter med outputlaget og beregner korreksjon lag for lag helt til det er i inputlaget. Videre oppdateres vektingene gjennom *gradient descent*, med hensyn på informasjonen om hvor mye vektene påvirker output-feilen. Dette bedrer prediksjonene fra modellen etter gjentatte kjøringer. (Veronica, 2021)

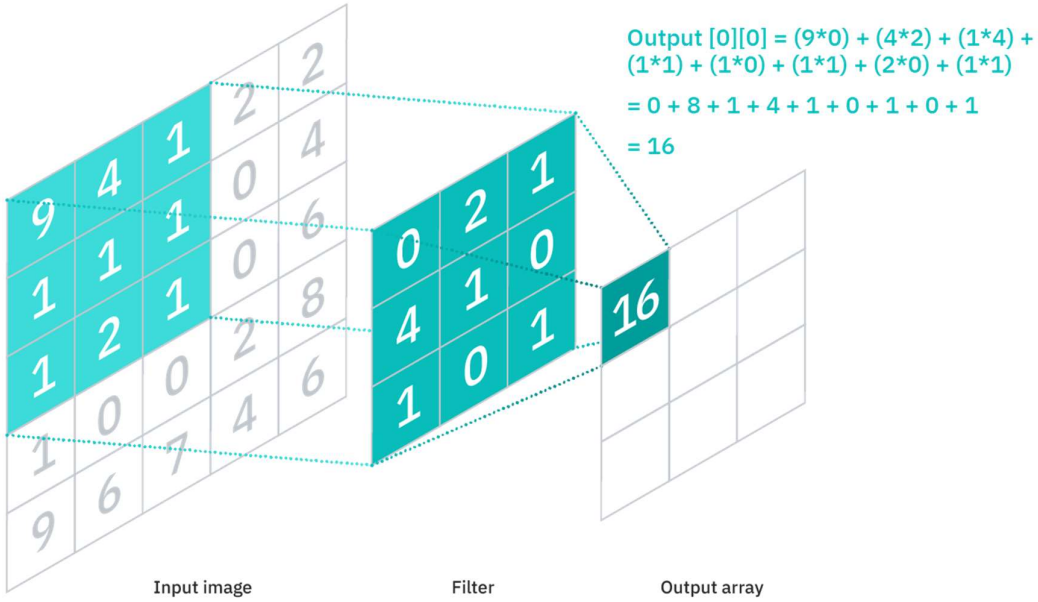
Forward propagation er et begrep for at data flyter gjennom et nevralt nettverk, fra inputdata til en prediksjon i utgangen av nettverket. Det foregår ved en tilpasset aktivering av nevronene i nettverket ut ifra inputdataens karakter. Denne aktiveringen skjer gjennom en aktiveringsfunksjon som summerer verdien til nevronene i forrige nevronlag multiplisert med vektingsverdien (Luhaniwal, 2019).

2.4.8 Convolutional Neural Network

Convolutional neural network (CNN) er en type neuralt nettverk som hovedsakelig blir benyttet til mønstergjenkjenning og er dermed en viktig del av oppgaven. CNNs gjør det mulig å detektere objekter uten at nettverket blir for krevende. Mønstergjenkjenning krever at man prosesserer store mengder data. Bilder inneholder mange piksler og hver piksel inneholder en inputverdi til nettverket. For et non-convolutional neural network vil da hver piksel være koblet til alle neuronene i neste lag. For et fargebilde med kun 64 piksler i horisontal og vertikal retning ville et neuron i det første skjulte laget inneha 12 288 vekter (64x64x3) (O'Shea & Nash, 2015). Dette fører til at kompleksiteten i utregningen blir mer og mer krevende for nettverket. Videre er non-convolutional neaural networks mindre robuste for endring av objektets posisjon i bildet, da hvert neuron kun aktiveres dersom bildet er i den seksjonen neuronet ligger i. For et non-convolutional neural

network gjør sammensetningen av nettverket at nødvendig prosesseringskraft er stor, mønstergjenkjenningen tar lang tid, nettverket tilpasser seg dårlig til endring av lokasjon på objekt og nettverket er følgelig mindre hensiktsmessig for oppgaven.

Et CNN løser problemet på en annen måte. Her blir påvirkningen på nærliggende piksler analysert gjennom et filter, som skal gjenkjenne ulik karakteristikk i bildet. Filtrene er matriser, med en størrelse på typisk 3x3 eller 5x5. (Stewart, 2019) Disse flyttes over bildet fra øverst i venstre hjørne til nederst i høyre hjørne. For hver del av bildet blir en verdi regnet ut ved hjelp av en matematiske operasjonen kalt konvolusjon. Et eksempel på konvolusjon kan sees i figur 2-10.



Figur 2-10: Konvolusjon. Input bildet ganges med et filter og får en output på en rute med verdi 16. Hentet fra (IBM Cloud Education, 2020).

Dette gjør at dersom man skal gjenkjenne for eksempel en vertikal kant, vil filteret i konvolusjonslaget gi oss en indikasjon på hvor tydelig kanten er i bildet, samt hvor mange ganger den er i bildet og hvor. Konvolusjon gir en fordel til neurale nettverk ved delvise koblinger. Der et konvensjonelt neuronnettverk er sammenkoblet med alle neuroner i neste og forrige lag, reduseres antall koblinger ved bruk av konvolusjonslag. På denne måten reduseres antall vekter, samt at lokasjonen av objektet i bildet heller ikke vil være av betydning. (Stewart, 2019)

Etter filtrene er dratt over bildet, dannes det er *feature map.* Dette tas gjennom en aktiveringsfunksjon for å bestemme hvorvidt det bestemte *feature'et* er på en gitt lokasjon i

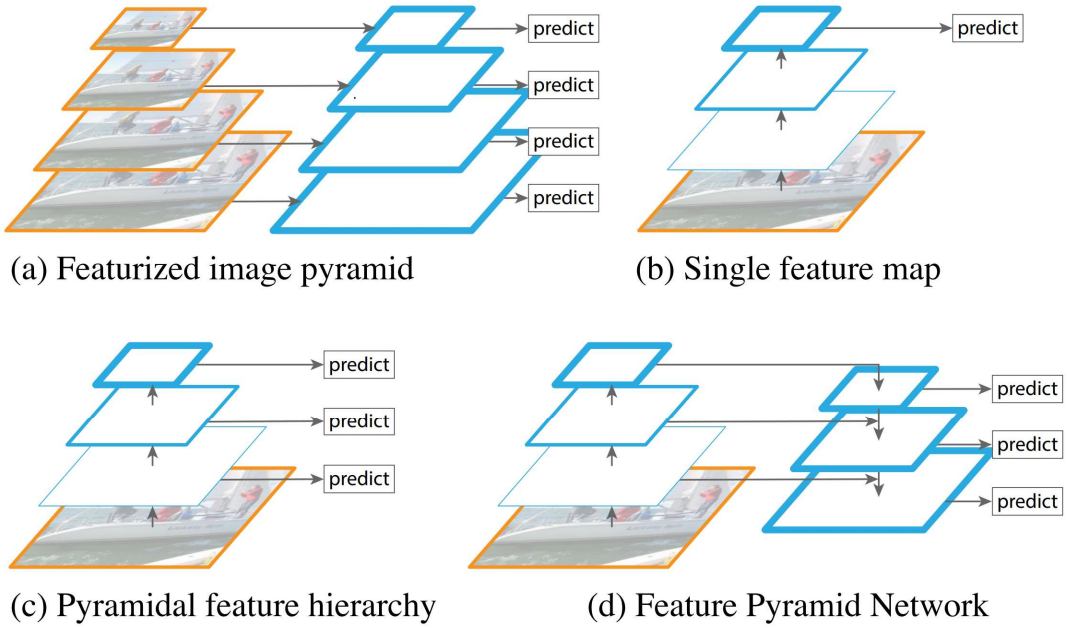
bildet eller ei. Den mest brukte aktiveringsfunksjonen er *rectified linear activation funcion*, forkortelse ReLU. Funksjonen setter output lik input dersom input er positiv, men dersom input er negativ vil output bli satt til null. Dette gjøres lag for lag i et CNN. De første lagene gjenkjenner gjerne kanter og hjørner. De midterste lagene gjenkjenner deler av objekter som for eksempel vertikal kant slik som i eksempelet over. De siste lagene gjenkjenner fulle objekter i forskjellige former og lokasjoner i bilder. Videre kan også *pooling*-lag implementeres i ett eller alle konvolusjonslagene. Disse lagene henter ut de største verdiene fra aktiveringsfunksjonen og sender verdiene videre til neste lag. Til slutt settes input fra de ulike filtrene inn i et *fully layered-lag,* som henter sannsynligheter og gir output av hvilke objekter som er i bildet. (Stewart, 2019)

Arkitekturen i CNN er oppsatt slik at det takler mønstergjenkjenning bedre. Hovedhensikten til arkitekturen er å minke antall parametere og overganger for å gjøre totalt antall matematiske operasjoner håndterbart. Input til et CNN er piksler fra et bilde. Nettverket benytter det matematiske konseptet konvolusjon på ett eller flere av lagene. Konvolusjon er hensiktsmessig for mønstergjenkjenning grunnet bruken av delvis tilkobling. (Stewart, 2019)

2.4.9 Feature pyramide network

Feature Pyramide Network (FPN) er en type nevralt nett som kan brukes til å øke prediksjonsevnen til en objektdeteksjonsmodell, ofte brukes et CNN som input i et FPN (Tsang, 2019). Denne typen nett fusjonerer features på en måte som øker gjenkjenningsevnen. Dette gjøres ved å skalere inputbildet i forskjellige oppløsninger som vist i figur 2-11 (a). Men det er en krevende oppgave å prosessere flere ulikt skalerte bilder samtidig. En mindre krevende måte å gjøre dette på er å trekke ut et feature-kart av de forskjellige oppløsningene på inputbildet, som vist i figur 2.11 (b), og fra hvert av disse feature-kartene med forskjellig oppløsning blir det gjort prediksjoner, som også vises i figuren. Alle disse prediksjonene fra forskjellig oppløsningsnivå dette blir benyttet i et FPN nett, her blir de kombinert med hverandre via en horisontal kobling på likt oppløsningsnivå og ovenfra og ned-kopling med et lavere oppløsningsnivå, som vist i figur 2-11 (d) (Tsang, 2019) Denne fusjoneringen av feature-maps i forskjellig oppløsning øker evnen til å oppdage mindre objekter, men beholder evnen til å oppdage større objekter. Mindre objekter

er lettere å oppdage ved høy oppløsning, mens større objekter er lettere å oppdage ved lavere oppløsning (Lin, et al., 2017)



Figur 2-11: Utviklingen av FPN (Lin, et al., 2017)

2.4.10 Transfer learning

Generelt er det lite hensiktsmessig å trene opp et fullstendig utrent nettverk hver gang man ønsker å kjenne igjen en ny objekttype og gjør transfer learning viktig. I vår oppgaven vil transfer learning bli benyttet til å videretrene et nettverk for å kjenne igjen vår egendefinerte klasse. Transfer learning benytter en eksisterende modell for en oppgave til å trene en ny modell for annen oppgave (Bozinovski, 2019). Modellen man tar utgangspunkt i er gjerne trent på et stort datasett, som typisk inneholder et bredt spekter av mønstergjenkjenningsoppgaver. Datasettet modellen er blitt trent på skal da være tilstrekkelig stort og generelt, slik at modellen kan fungere som en generisk modell for den visuelle verden. Dette gjør at den videretrente modellen kan bruke den lærte modellen og dens *feature map* som utgangspunkt. Fordelen dette gir er at den nye modellen ikke trenger å trenes på et stort datasett for å kunne gjøre prediksjoner for sin oppgave (TensorFlow, 2022)

Til nå har denne oppgaven tatt for seg grunnsteinene til prosjektet. I denne oppgaven vil spesielt CNNs og FPNs bli implementert til å gjenkjenne egendefinert klasse gjennom

transfer learning av nettverkene. Dette fører oppgaven inn på maskinlæring for objektdeteksjon for å gi en forståelse for hvordan maskinlæring benyttes i fagfeltet.

2.5 Maskinlæring for objektdeteksjon

Objektdeteksjon er et fagfelt innenfor computer vision for å lokalisere objekter i bilder eller videoer. For å lokalisere objektene benyttes oftest maskinlæring for å gi meningsfulle resultater (MathWorks Inc, u.å.). En type dyplæring som blir mye brukt til objektdeteksjon er CNN og beskrives i 2.4.8. For å utvikle et CNN for eget bruksområde finnes det to fremgangsmåter:

1. Lage og trene egen modell.
2. Bruke en ferdig trent modell.

Dersom man lager egen modell, må arkitekturen i nettverket designes til å gjenkjenne objektene systemet skal gjøre prediksjoner på. Dette krever veiledet maskinlæring på store datasett for et trene opp et slikt CNN. Tilnærmingen kan gi gode resultater, men er også svært tidkrevende. Den alternative løsningen er da å fin-tune en pretrent modell for et annet formål. Dette gjøres gjennom transfer learning.

2.6 Objektdeteksjon på edgeenheter

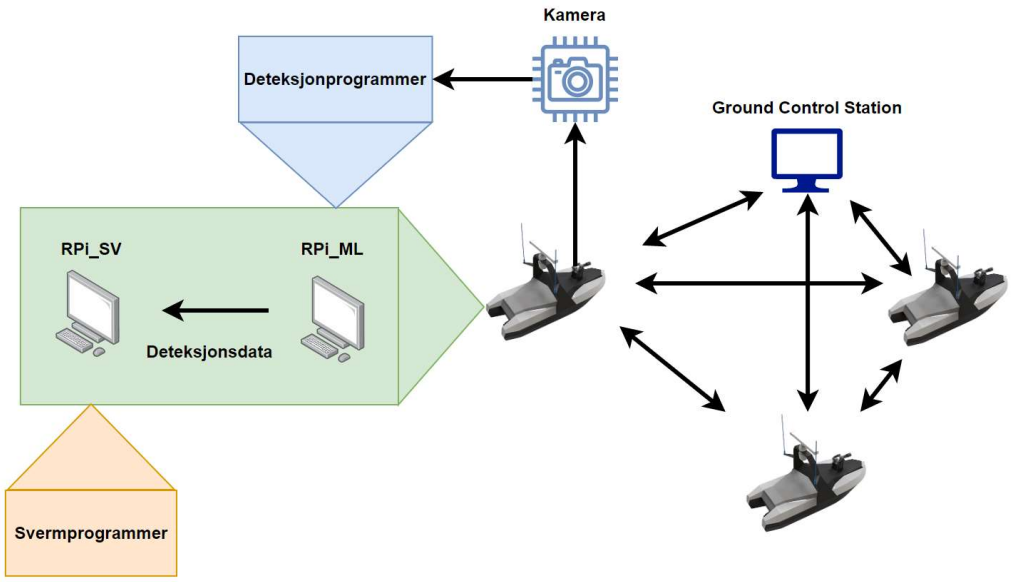
Selv om den teknologiske utviklingen har vært stor innen objektdeteksjon de siste årene, så krever fortsatt mange av modellene mer prosessorkraft enn hva mobile- og edgeenheter tilbyr. Ofte når man bruker mobile- og edgeenheter så sendes dataen som skal prosesseres til en server som er mye kraftigere og laget spesifikt for denne typen oppgaver (Chen & Ran, 2019). Alternativet er at dataen blir prosessert lokalt på selve edgeenheten. Dette gir muligheter for nærmere real-time prosessering og man er ikke avhengig av å være tilkoplet et nettverk, men dette fordrer også en særs effektiv maskinlæringsmodell som ikke krever for mye prosessorkraft. (Chen & Ran, 2019)

En edgeenhet må i utgangspunktet ha de samme maskinvarekomponentene som en vanlig datamaskin, men hver enkelt komponent er vanligvis mye mindre. Og når størrelsen til komponentene minskes så minskes ofte også ytelsen. Noen komponenter er ofte utelatt for å spare plass, eksempelvis dedikerte grafikkprosessorer. Tilgangen på strøm er også en begrensende faktor på edgeenheter, da de ofte benytter et lite batteri for å gi strøm til

systemet. Dermed er også prosessorene utviklet for å være strømbesparende noe som går på akkord med høy ytelse. Mindre enheter har ofte også mindre lagringsplass, noe som kan gjøre at moderne objektdetektosjonsmodeller ikke får plass på minnet. Begrensningene til en edgeenhet som skal brukes til objektdeteksjon er dermed oppsummert ved prosessorkraft, arbeidsminne, lagringsminne og strømtilførsel. (Chen & Ran, 2019)

3 Implementering

I teoridelen har vi lagt grunnlaget for å forstå resten av oppgaven. Prosjektet skal benytte objektdeteksjon på dronesverm til å gjøre deteksjoner og identifikasjoner av fartøy. Ved å benytte data fra åpen-kilde deteksjonsmodeller skal dronesvermen kunne øke sin situasjonsforståelse. For å sette sammen et svermsystem for objektdeteksjon er det flere komponenter som samhandler. Første komponent er dronene med tilhørende svermprogramvare. Dronene er tilgjengelige på skolen, og svermkoden kunne hentes ut fra GitHub (Hellesnes & Lyssand, 2019). Denne delen av systemet kjører svermalgoritmen og vil senere bli omtalt som RPi_SV (Raspberry Pi Swarm). Andre komponenten av systemet skal kunne detektere fartøy og sende nyttig deteksjonsdata til RPi_SV. Datamaskinen som kjører denne delen av systemet kalles RPi_ML (Raspberry Pi Machine Learning) i resten av oppgaven. Deteksjonsdataen som deles med RPi_SV skal gi informasjon om deteksjon og lokalisering.



Figur 3-1: Systemstruktur (båter hentet fra Challenger Aerospace Systems, 2019)

Figur 3-1 viser hvordan systemet er satt opp. Hver drone inneholder nå et kamera og en ekstra datamaskin som kjører deteksjonsprogrammene. Deteksjonsprogrammene skal detektere fartøy med kameraet som sensor, lokalisere det med relativ peiling og kommunisere dette videre til RPi_SV. RPi_SV kommuniserer videre med resten av svermen samt

operatørpanelet, kalt Ground Control Station i figuren. Relativ peiling er peilingen detektert objekt har i forhold til peilingen på dronen. De ulike delene av systemet vil bli videre forklart i resten av kapitelet.

3.1 Skroget med tilhørende maskinvare

Dronene trenger et skrog og skrogets design er videreført fra bacheloroppgaven om plattform dronesverm og vises i figur 3-2 (Hellesnes & Lyssand, 2019). Det inneholder tre motorer for fremdrift samt tre ror for styring. Motorene er markert med blått og rorene er markert med rødt. Skroget inneholder også en Raspberry Pi, en Pixhawk 4 (ikke vist her) samt en Arduino. Raspberry Pi og Arduino er festet på toppdekselet i baugen, makert med grønt. Disse delene av skroget er allerede implementert og vil ikke være essensiell for oppgaven videre. Skroget videreutvikles ved at vi tilfører ytterligere en Raspberry Pi-enhet for å kunne kjøre objektdeteksjonsmodellen og deteksjonsprogrammene. Vi har også montert et RPi-kamera på dronen for å kunne drive objektdeteksjon.

Figur 3-2: Skroget til hver enkelt drone (Sauter, 2019)

3.1.1 Kamera og kameraholder

For å kunne kjøre live objektdeteksjons-algoritmen, benyttes Raspberry Pi Camera V2 som optisk sensor. Kameraet har lav vekt og kommuniserer godt med RPi. Figur 3-3 viser et bilde av kameraet som er koblet til en RPi. Kameraet er festet i baugen på dronen for at skroget ikke skal komme i kameraets synsvinkel. Videre har kameraet en horisontal synsvinkel på 62.2 grader (The Raspberry Pi Foundation, 2016). Synsvinkelen bruker vi til å regne relativ peiling på deteksjon i forhold til dronens peiling. For å kunne gjøre dette kalibrerer vi dataen fra RPi kamera og objekdeteksjonsmodellen. Måten dette har blitt løst på vil bli gjennomgått i 3.6. For å holde kameraet på plass har vi benyttet oss av en kameraholder.

Figur 3-3: RPi Camera tilkoblet en RPi. Hentet fra (Elfa Distrelec, 2022)

For å implementere RPi-kamera på skroget er det satt på et kamerahus i fronten av båten. Nøyaktighet under påsettelse av kamerahusets er essensielt, da relativ peiling skal komme i forhold til dronens kjøreretning. Feilvinkling av kamerahuset kan gjøre at deteksjoner blir peilet feil i forhold til dronene. Kamerahuset er 3D-printet og er vist i figur 3-4.

Figur 3-4: Kamerahuset til kameraet.

3.1.2 Raspberry Pi

I dette prosjektet benytter vi to Raspberry Pi datamaskiner per svermenhet som vist i figur 3-5. RPi_SV styrer svermalgoritmen og RPi_ML installeres for å håndtere objektdeteksjon av andre fartøy gjennom kameraet. Vi benytter oss av både Raspberry Pi 4 og 3b+ til å drive objektdeteksjon for å belyse forskjeller i prestasjon ved forskjellig prosessorkraft. Begge RPiene trekker strøm via en DC/DC omformer som er inkludert i en av motorkotrollerne på dronen.

Figur 3-5: Raspberry Pi 3B+. Hentet fra (Elfa Distrelec, 2022)

Vi undersøkte muligheten for å bruke RPi_SV til å kjøre både svermprogrammene og deteksjonsprogrammene, men det gikk ikke. På RPi_SV var det installert et operativsystem som ikke var kompatibelt med maskinlæringsplattformen vi skulle benytte og valgte derfor å benytte oss av to Raspberry Pi's. Dette førte til at datamaskinene på hver drone måtte kommunisere med hverandre og gjorde kompleksiteten i systemet større. Likevel kan det sies å være en mer robust løsning. RPi er ikke en kraftig datamaskin og det er dermed plausibelt å anta at kjøring av to krevende programmer på samme datamaskin ville vært hemmende for prestasjonen til systemet som helhet. Videre viste det seg at å kjøre prosessene på to ulike datamaskiner var nyttig i testing og utvikling av systemet.

3.2 Programvaremiljøet for objekt deteksjon

Ettersom at deteksjonsprogrammene ikke kunne kjøres på samme RPi som svermprogrammene måtte vi finne et passende programvaremiljø for RPi_ML. Valget av operativsystem for RPi_ML endte dermed på Raspbian 10 buster, som er det anbefalte operativsystemet å kjøre maskinlæringsplattformen vi har valgt.

Valget av maskinlæringsplattform ble TensorFlow Lite (TFLite). TFLite er en plattform som er enkel å anvende, har mange ferdigutviklede maskinlærings-modeller tilgjengelig og det er godt tilrettelagt for å videreutvikle disse. Lite-utgaven passer bra for Raspberry Pi 4 og 3b+ da dette er en enkeltbretts-datamaskin med begrenset prosessorkraft og lokal lagringsplass. Denne utgaven er tilpasset edgeenheter og tilbyr dermed en maskinlæringsmodell som er presis og effektiv for RPi_ML (TensorFlow, 2022). Vi har kopiert TF sin

kildekode for å kjøre objektdeteksjonmodeller (TensorFlow, 2022). I tillegg har vi lagt til funksjoner i kildekoden som sørger for at vi kan hente ut data som benyttes til å peile objektet og en funksjon for å dele deteksjonsdata med svermen og operatørpanelet.

3.3 Maskinlæringsmodell og trening

Valget av maskinlæringsmodell til å gjøre deteksjoner for dronesvermen er Efficient-Det_Lite (Tan, Pang, & Le, EfficientDet: Scalable and Efficient Object Detection, 2019). Denne modellen er skalert opp i fem forskjellige skaleringsversjoner som krever økende grad av prosessorkraft. De ulike versjonene er vist i tabell 2. Her ser vi at større modeller gir bedre mAP, men at forsinkelsen også øker. Derfor må man gjøre en avveiing om hva som er viktigst for prosjektet.

Som nevnt i tabell 2 et av målene i oppgaven å kunne detektere egendefinert klasse med presisjon høyere eller lik 80%. Dette setter krav til hvilken en tilstrekkelig god skaleringsversjon av EfficientDet_Lite for videretrening. Samtidig skal maskinlæringsmodellene kjøres på henholdsvis RPi 4 og 3b+ i dette prosjektet. Det setter en begrensning for tilgjengelig prosessorkraft som fører til forsinkelse under kjøring av modellen. Valget av skaleringsversjon ble gjort på grunnlag av prøving og feiling.

Modell	Størrelse	Forsinkelse (RP1 4)	mAP
EfficientDet_Lite0	4.4 MB	146 ms	25.69%
EfficientDet_Lite1	5.8 MB	259 ms	30.55%
EfficientDet_Lite2	7.2 MB	396 ms	33.97%
EfficientDet_Lite3	11.4 MB	726 ms	37.70%
EfficientDet_Lite4	19.9 MB	1886 ms	41.96%

Tabell 2: Hentet fra Tensorflow (Abadi, et al., 2015)

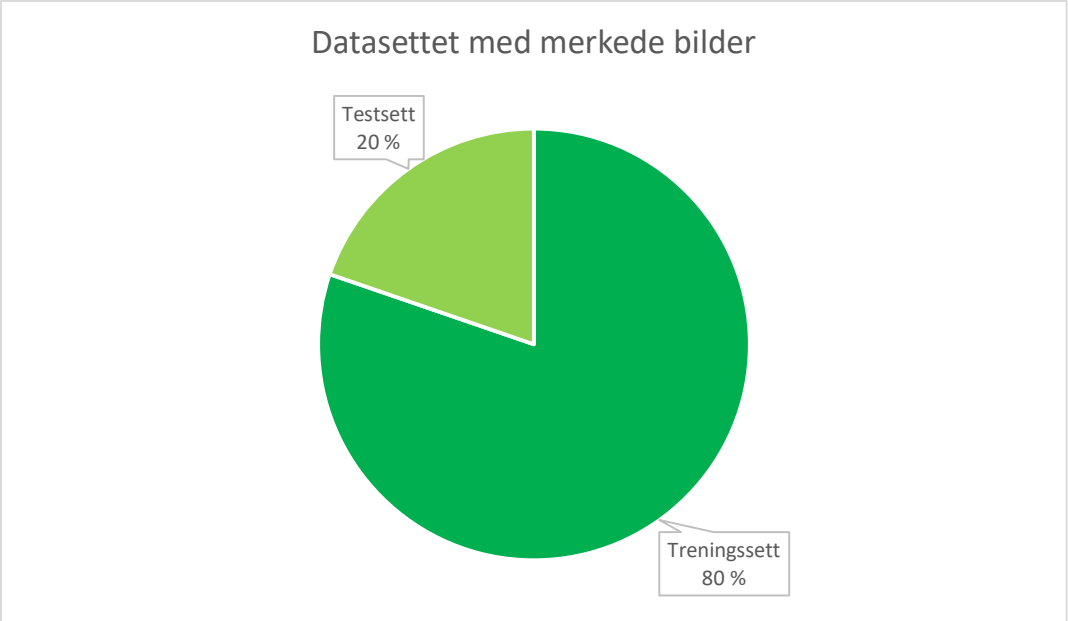
Det kom frem at EfficientDet_Lite1 var best egnet for RPi 4 og at EfficientDet_Lite0 var best egnet for RPi 3b+. Dette er de to minst krevende skaleringsversjonene og ble valgt fordi de kan kjøres relativt raskt på våre datamaskiner, noe som var veldig nyttig under testing og utvikling av dette prosjektet. Grunnen til at valget falt på to forskjellige skaleringsversjoner var at vi benytter både RPi 4 og RPi 3b+ som RPi_ML. RPi 4 er en kraftigere datamaskin enn RPi 3b+, som gjorde at den kunne kjøre en modell med bedre mAP.

EfficientDet_Lite er i utgangspunktet trent på datasettet «MS COCO» (Lin, et al., 2014) som har mer enn 300.000 bilder og inkluderer 80 klasser av vanlige objekter, blant annet objektet «ship». For å spare mye tid har vi benyttet metoden *transfer learning,* der man videretrener en modell som allerede har trente vekter.

Som del av oppgaven ble det definert to nye klasser som detektoren skal gjenkjenne, en klasse som simulerer en fiendtlig fartøysklasse «Svetlana Class» og en klasse som er andre vennlige svermenheter «Swarm Unit». Valget av to modeller er gjort for å lære modellen å skille mellom Svetlana klassen og andre fartøyer. Det kan i tillegg være nyttig for sverm-enhetene å gjenkjenne hverandre visuelt med tanke på en eventuell videreutvikling av svermsystemet.

Figur 3-6: Eksempelbilde fra treningsdatasettet, på venstre Svetlana Class og Swarm Unit på høyre siden.

Til å videretrene modellene ble det tatt 223 bilder som viser de to nye klassene, bildene ble tatt med varierende vinkel på objektene og med varierende avstand. Et eksempel er vist i figur 3-6. Deretter drev vi veiledet maskinlæring ved å markere objektene i programmet labelImg som lager en xml-fil med pikselkoordinater for hvert objekt. Så delte vi opp datasettet med bilder og xml-filer i et treningssett og et testsett. Som vist i figur 3-7.



Figur 3-7: Viser fordelingen av data for *transfer learning*

Treningen av modellen foregikk i en *Google Colabs notebook* der vi brukte TensorFlow sin kildekode (Abadi, et al., 2015). Der lastet vi opp treningsdatasettet og testdatasettet med tilhørende filer som har pikselkoordinater for objektene. Vi valgte Efficientdet_lite1 og Efficientdet_lite0 som modeller å videretrene. For videretreningen valgte vi at alle vektene i modellen skulle trenes, fordi det kan øke prediksjonsevnen. (Abadi, et al., 2015)

Vi satte treningsepoker til 20, som gjør at den går igjennom treningsdatasettet 20 ganger. Batch size ble satt til 4, dette angir hvor mange bilder som sendes til nettverket om gangen. Med batch size 4 tok det 47 skritt å gå igjennom alle bildene i treningssettet, og for hvert av disse skrittene ble vektene i nettverket justert. Tabell 3 viser parameterne vi valgte i treningen av modellen. Valgene er basert på lignende eksempler samt prøving og feiling. Senere i prosjektet ble også EfficientDet_Lite4 videretrent, for å undersøke hvorvidt en modell med høyere utgangspunkt i mAP gir markant bedre prediksjonsnøyaktighet etter videretrening. Navnet på de videretrente modellene ble Svetlana_Detector, med 3 forskjellige versjonsnummer.

	Svetlana_Detector0	Svetlana_Detector1	Svetlana_Detector4

Modell	Efficientdet_lite0	Efficientdet_lite1	Efficient-det_lite4
Batch size	4	4	4
Train_whole_model	true	true	true
Epochs	20	20	20
Antall treningsbilder	179	179	179
Antall testbilder	44	44	44

Tabell 3: Tabellen viser parameterne vi valgte i treningen av modellen.

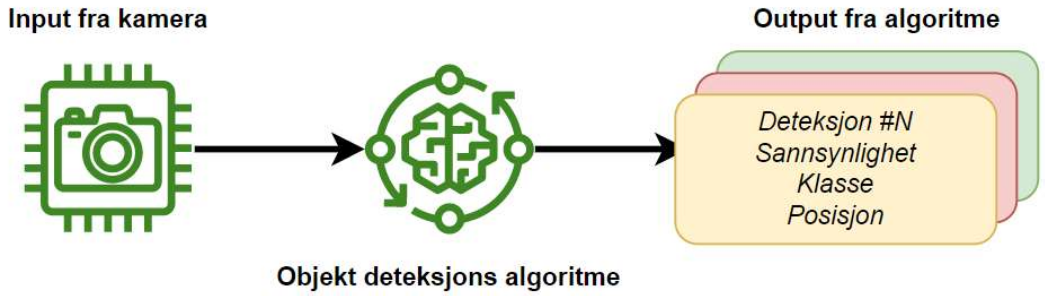
3.4 Hovedprogrammet og filstrukturen

Kapittelet har til nå tatt for seg hvilken deteksjonsmodell vi har videretrent, og fører oppgaven videre inn i hvordan vi har implementert deteksjonsmodellen i programvaren. Deteksjonsprogrammene og deteksjonsmodellen kjøres gjennom scriptet *detect.py* (TensorFlow, 2022). Koden er hentet fra TensorFlow under en Apache 2.0 lisens. Programmet klassifiserer mulige objekter og lokaliserer dem i pikselformat gjennom videofeeden fra kameraet. Disse klassifikasjonene blir predikert ut ifra de klassene modellen er trent på (TensorFlow, 2022). Målet i vår oppgave er at Svetlana-klassen skal bli klassifisert. Dersom et bilde med kjent objekt blir sendt gjennom en deteksjonsmodell vil output gjengi følgende om objektet:

- Objekts klassifikasjon
- Boksen rundt objektet
- Sannsynlighet for at det er objektklassen

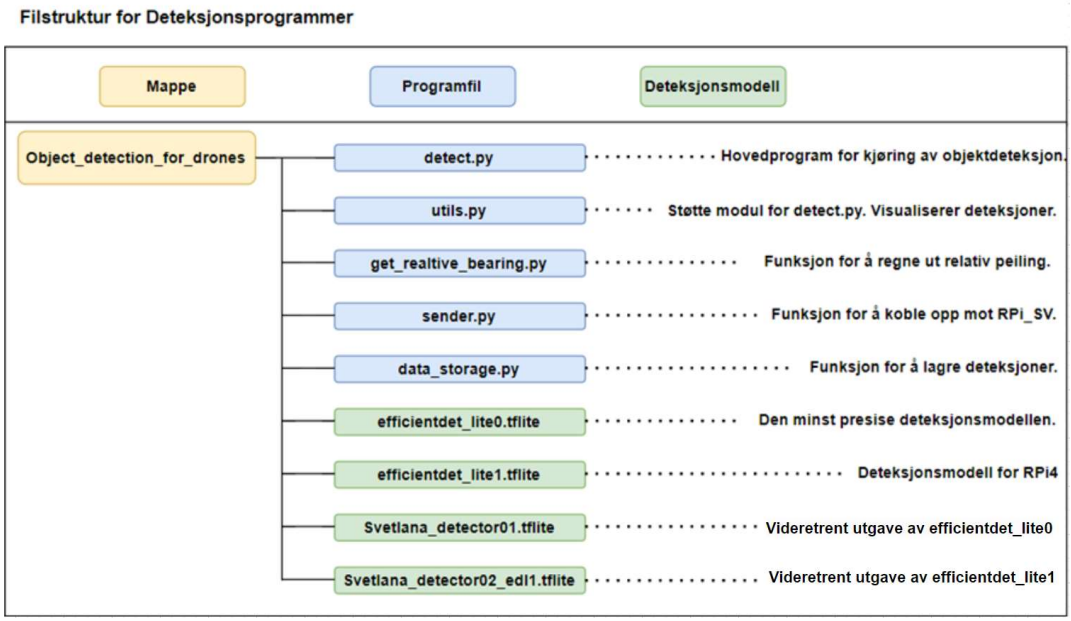
Disse gjengis som *array's* i systemet. *Arrayene* spenner fra indeksene 0-3 der indeks 0 tilhører lokasjon. Lokasjonen gjengir inn-boksingen av objektet. Disse verdiene gjengis på følgende format [topp, venstre, bunn, høyre]. Indeks 2 gjengir sannsynligheten for at objektet er klassifisert rett. Her gis prediksjonen en score på mellom 0 og 1. Indeks 1 er *arrayen* tilhørende klassen og identifiserer hva objektet er blitt klassifisert som. Indeks 3 er antall deteksjoner, N, som er funnet i bildet. Hver deteksjon blir da nummerert fra 0 til N deteksjoner. Indeksene fra 0 til 2 beskriver den enkelte deteksjon i indeks 3. Figur 3-8

visualiserer hvordan dette fungerer. Programmene vi har utviklet bruker denne informasjonen til å kunne detektere, lokalisere og kommunisere fartøysdeteksjon samt lagre bilder for videretrening av modellen.



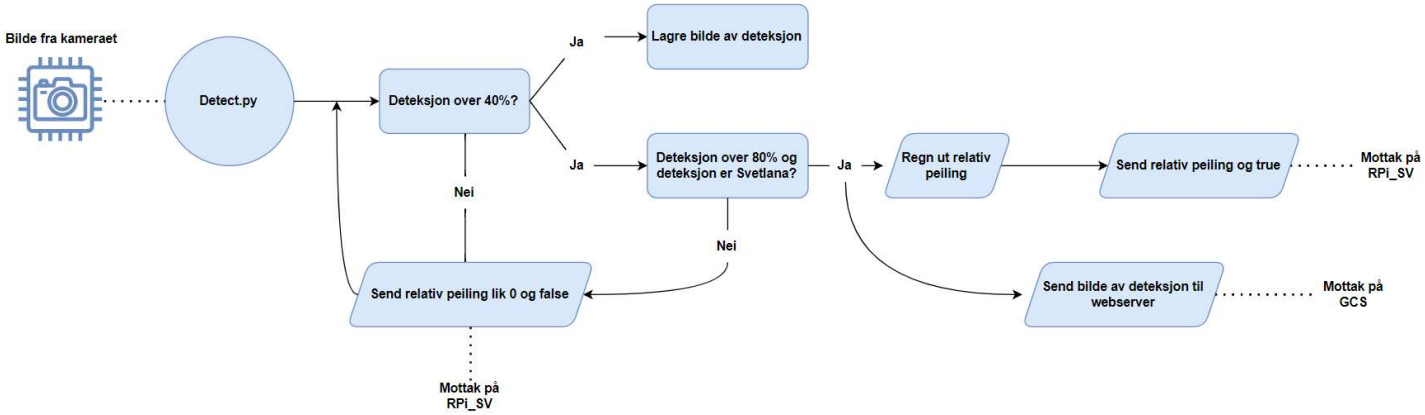
Figur 3-8: Objektdeteksjonsvisualisering

Filstrukturen til deteksjonsprogrammene består av en mappe der alle programmene og deteksjonsmodellene ligger. Strukturen vises i figur 3-9. Det er programmet *detect.py* som kjøres for at objektdeteksjonen skal fungere. I dette scriptet hentes den modellen man ønsker å benytte. For denne oppgaven er det videreutviklet de to modellene som er nederst i figuren. Programmet er utviklet av TensorFlow og modifisert av oss. Scriptet er modifisert til å hente funksjoner fra våre script og koble dem sammen for å kunne regne ut relativ peiling, lagre deteksjonsdata samt sende deteksjonsdata til RPi_SV.



Figur 3-9: Total oversikt over filstruktur på RPi_ML

Hovedprogrammet kjører deteksjonsmodellen kontinuerlig og kjører funksjonene dersom kriteriene for at det skal kjøres er oppnådd. Programflyten i figur 3-10 viser hvordan programmet er modifisert til å fungere for hvert bilde som kommer fra kameraet.



Figur 3-10: Programflyt for *detect.py*

3.5 Lokal lagring av data til videretrening

Lokal lagring gjøres internt på hver enhet. Mer spesifikt vil RPi_ML lagre deteksjoner som har en sannsynlighet over 40%. Dataen lagres slik at modellen kan videretrenes ved hjelp av *transfer learning*. Idet *detect.py* blir initiert vil scriptet automatisk opprette en ny mappe. Nummereringen av mappene er avhengig av hvor mange ganger *detect.py* er kjørt. Etter opprettelse av ny mappe er strukturen klar for å skrives til. Programmet vil da skrive til mappen dersom en deteksjon er gjort med sannsynlighet over 40%. Ved deteksjon sender programmet inn bildet av deteksjonen. Dette er representativt for hva kameraet ser og vil være tilstrekkelig for videretrening. Bildene av deteksjonene settes i sammenheng med en loggfil. Loggfilen blir opprettet ved første deteksjon og inneholder følgende informasjon om deteksjonene:

- Klasse
- Sannsynlighet for at det er deteksjonen
- Navnet på bildet som er tilhørende deteksjonen

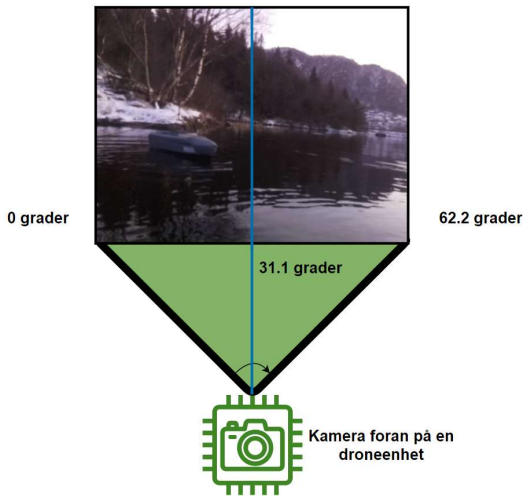
Navnet på bildene er nummerert i kronologisk rekkefølge. Nummereringen starter på null og øker for hver gang det er blitt gjort en deteksjon på over 40%. Mappestruktureringen, bildelagringen og loggfilen danner grunnlaget for å kunne videretrene modellen på eget datasett.

3.6 Relativ peiling

Et av hovedmålene for oppgaven har vært å kunne lokalisere fartøy. Enhetene skal internt kunne regne ut relativ peiling ved hjelp av kamera og deteksjonsmodellen. Relativ peiling betegnes, i denne oppgaven, som peilingen detektert fartøyet har i forhold til dronens kjøreretning.

I *detect.py* regnes relativ peiling ut dersom en deteksjon er av rett klasse og at sannsynligheten er over 80%. Dette gjøres gjennom en egendefinert funksjon i *get_relative_bearing.py*. Funksjonen kobler outputdata fra maskinlæringsalgoritmen, kameraets synsvinkel, samt tolkning av piksler i horisontal retning sammen for å regne ut relativ peiling.

Funksjonen henter bredde på boks samt hvor boksen starter fra visualiseringsfunksjonen *utils.py*. Dette er data som ikke gir mening for hver enkelt drone. Dronene vet hvilken retning som er frem, men ikke at kameraet er rettet fremover. Av den grunn er det viktig å definere midtpunktet i kameraet. Kameraet på denne enheten har en synsvinkel på 62.2 grader i horisontal retning. Dronens peiling vil da tilsvare 31.1 grader inn i kameraets synsvinkel. Dette illustreres i figur 3-11.



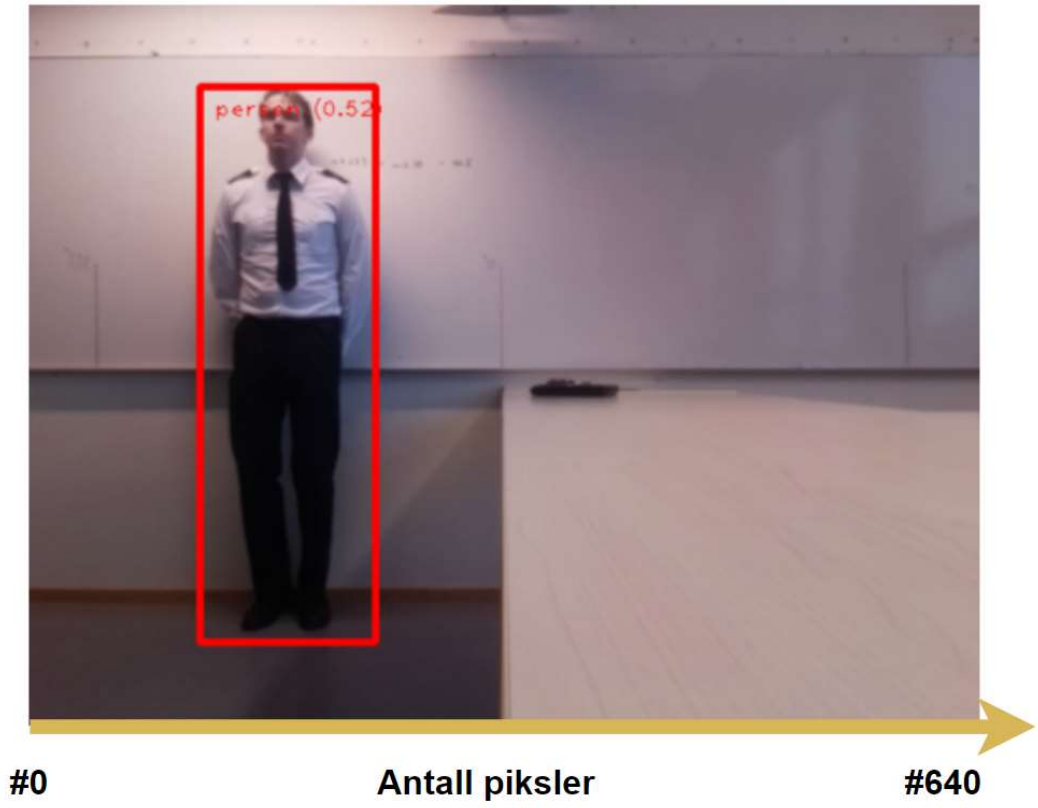
Figur 3-11: Kameraets synsvinkel opp mot dronens peiling

Dette punktet settes da som nullpunkt. Dette er ikke data RPi_ML bruker for å lokalisere objektet i bildet. Maskinlæringsalgoritmen tolker piksler for å sette boksen for objektet. Av den grunn må pikslene settes i sammenheng med synsvinkelen til kameraet.

Programmet *detect.py* bruker totalt 640 piksler i horisontal retning som input fra kameraet. Settes dette i sammenheng med den totale synsvinkelen til kameraet, kan vinkel per piksel regnes ut. Dette gjøres ved å dele total synsvinkel på antall piksler i horisontal retning, slik som i formelen under:

$$Vinkel\ per\ piksel = \frac{total\ synsvinkel}{totalt\ antall\ piksler}$$

Figur 3-12 viser et eksempel på en korrekt innbokset deteksjon. I denne figuren kommer det også frem hvordan pikslene øker fra 0 til 640.



Figur 3-12: Visualisering av antall piksler i horisontal retning

Denne dataen må videre tilpasses slik at den kan tolkes i sammenheng med dronens posisjonsdata. Som nevnt tidligere vil peilingen være lik 31.1 grader fra start av synsvinkel. For maskinen tilsier det 320 piksler inn i bildet. Videre må relativ peiling på objekt ses i

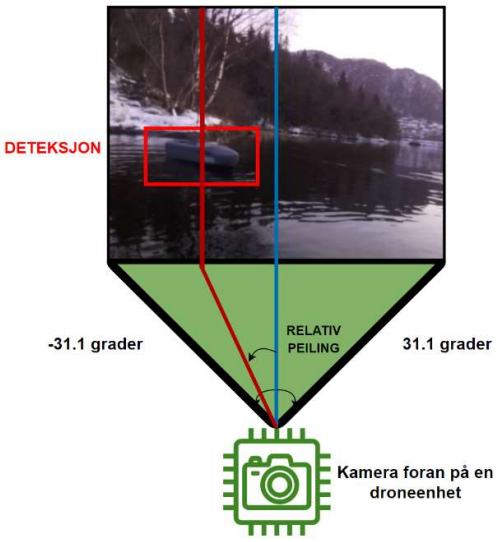
sammenheng med dronens forståelse av peiling. Peilingen øker når dronen går mot styrbord og minker når dronen går mot babord. Av den grunn blir uttrykket for relativ peiling som følger:

$$Relativ\ peiling\ (i\ piksler) = Kamera\ senter - (\frac{Bredde\ på\ boks}{2} + start\ av\ boks)$$

Settes de to formlene sammen ender vi opp med følgende:

$$Relativ\ peiling\ (i\ grader) = (vinkel\ per\ piksel) * (relativ\ peiling(i\ piksler))$$

På denne måten tolkes dataen og gir relevant output for hver drone samt svermen i helhet. Figur 3-13 viser hvordan RPi_SV vil tolke dataen som kommer fra RPi_ML. Blå linje i midten viser faktisk peiling. Dronen kan se fra -31.1 grader til 31.1 i forhold til egen peiling. I figuren er det satt opp en fiktiv deteksjon på babord side som blir lokalisert ved hjelp av relativ peiling.

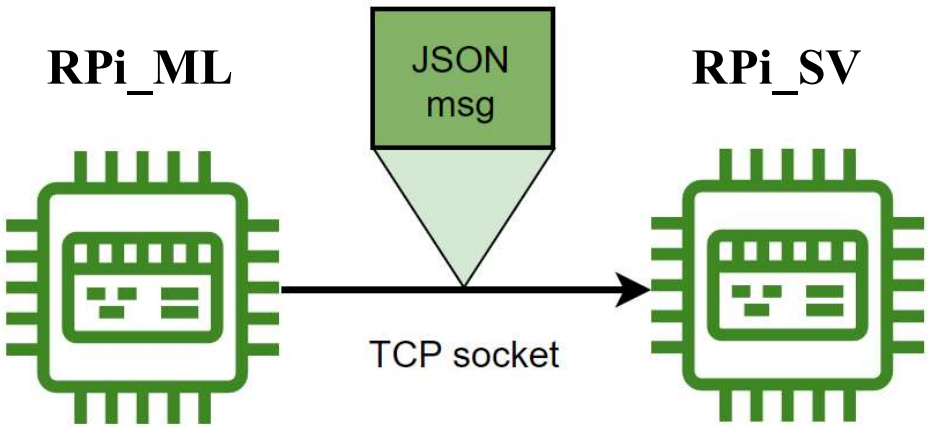


Figur 3-13: Visualisering 38elative peiling

3.7 Kommunikasjon av deteksjonsdata

For å sende relativ peiling fra RPi_ML til RPi_SV på en effektiv måte er det satt opp TCP-kommunikasjon. Dette gjøres ved å sette opp RPi_SV som server og RPi_ML som klient. RPi_SV setter opp serveren på sin IP-adresse på port 9091. En visualisering av dette vises i figur 3-14. Her lytter RPi_SV på hva som kommer inn til serveren fra klienten RPi_ML. IP-adressene er også koblet opp internt mellom RPi'ene, slik at RPi_ML sender til RPi_SV om bord på samme droneenhet. RPi_ML sender relativ peiling sammen med en boolsk verdi. Den boolske variabelen benyttes for å fremheve hvorvidt deteksjonen er av interesse eller ikke. Denne dataen vil være lik null og false dersom det er Svetlana og

sannsynlighet større eller lik 80% for at det er klassen. Dersom programmet får inn deteksjon på Svetlana, vil RPi_ML sende true med 39elative peiling I forhold til dronens peiling. Dataen sendes I JSON-format og vil videre bli konvertert slik at det kan bli 39elative39er I svermens datastrøm på RPi_SV.

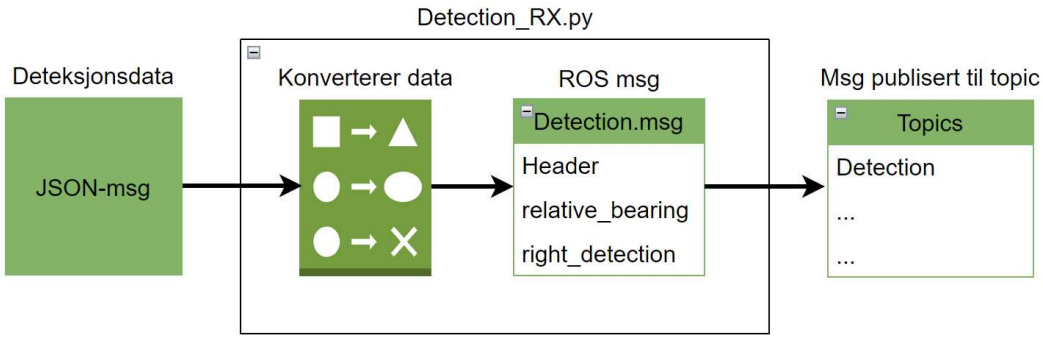


Figur 3-14: Modell for kommunikasjon mellom RPi-enhetene

For å implementere relativ peiling i datastrømmen er det opprettet et nytt hovedprogram på RPi_SV. Hovedprogrammet heter *Detection_RX* og vil suppleres til strukturen for kommunikasjon i originalprogrammet. Originalprogrammet har allerede to hovedprogrammer som skal ta seg av kommunikasjonen mellom enhetene. Henholdsvis er dette programmer som styrer sending og mottak av data. Senderen henter data fra hver enhet og sender dette som en multicast-melding. Mottakeren tar imot disse meldingene og skriver data fra meldingene til delsystemene i ROS (Hellesnes & Lyssand, 2019). Innlagt i denne strømmen er det lagt til et nytt *topic* for deteksjonsdata fra RPi_ML. Hovedprogrammet fungerer derfor som en sensor som kontinuerlig oppdaterer deteksjonsdata på RPi_SV. Dette fører til ytterlige implementeringer i sender- og mottakerprogrammene.

Mottaker av deteksjon konverterer data fra JSON-format til ROS-format slik at den kan nyttes i systemet for dronesvermen. JSON-meldingen fra RPi_ML blir omgjort til en *message* og videre publiseres denne meldingen på deteksjons *topic'et*. På denne måten er deteksjonsdataen til disposisjon for andre programmer i systemet. *Deteksjon_RX* er hovedprogrammet for denne prosessen, men støtter seg på undermodulen *Detection_input*. Hovedprogrammet initierer ROS-noden og kjører koden i *Detection_input*-scriptet. I denne undermodulen blir serveren satt opp, dataen konvertert til ROS *message* og videre

publisert til *topics*. Figur 3-15 beskriver hvordan dataen blir håndtert og videreformidlet til ROS kommunikasjonsflyt.



Figur 3-15: Konvertering av JSON-melding til ROS topic

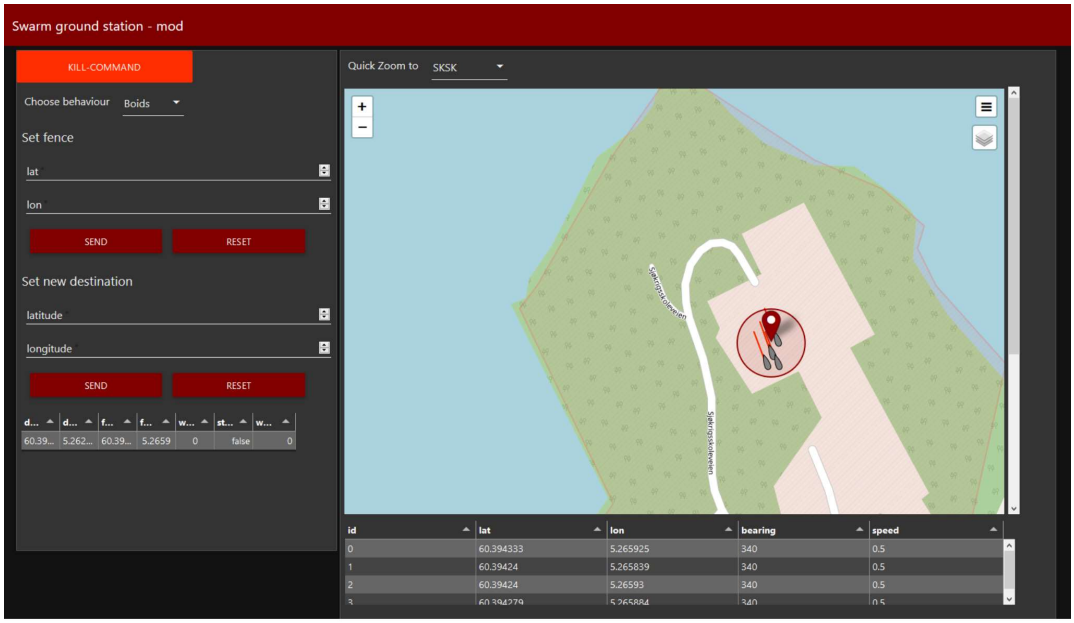
Implementeringen av deteksjonsdata i kommunikasjonsflyten skal tilføye et nytt *topic* til programmene. I begge implementeringene følger vi samme fremgangsmåte som i originalprogrammet. Denne kommunikasjonen er avhengig av å hente data fra *topic's* og derfor må deteksjonsdata publiseres der. Publiseringen til deteksjons-*topic'et* er allerede implementert og de andre *nodene* kan dermed abonnere på *topic'et*. *Topic'et* blir abonnert på av senderen til RPi_SV og kan dermed mottas av mottakeren blant de andre enhetene i svermen.

3.8 Brukergrensesnitt

Operatørpanelet har ikke vært hovedfokus for utviklingen av dronesvermkonseptet vårt. Likevel blir menneske-maskin-interaksjonen viktig for å dra nytte av konseptet for maritim overvåkning. Uten en mulighet for å observere hva som blir overvåket eller detektert, ville ikke konseptet bedret situasjonsforståelsen for andre enn svermen selv. Dersom konseptet skal kunne nyttiggjøres i en sjømilitær kontekst er derfor brukergrensesnittet viktig.

Brukergrensesnittet må ha oversikt over hvor svermen er og kunne tilby operatør nyttig deteksjonsdata fra hver enkelt drone. Operatørpanelet fra bacheloroppgaven om sverm fra 2019 er videreført i vår oppgave (Hellesnes & Lyssand, 2019). Her kan operatør observere hvor svermen er samt styre hvor operasjonsområdet til svermen skal være. For å bygge videre på dette har vi modifisert operatørpanelet til å kunne vise deteksjonsdata fra hver enkelt drone. Dette fører til at operatør kan overvåke bildebyggingen svermen gjør fra panelet.

Brukergrensesnittet kan styre dronesvermen ved å flytte operasjonsområdet rundt på kartet, sende et kill command for å legge dronesvermen død i vannet, velge sverm-atferd og styre svermen mot en spesifikk destinasjon. Figur 3-16 viser brukergrensesnittet før vi videreutviklet det.



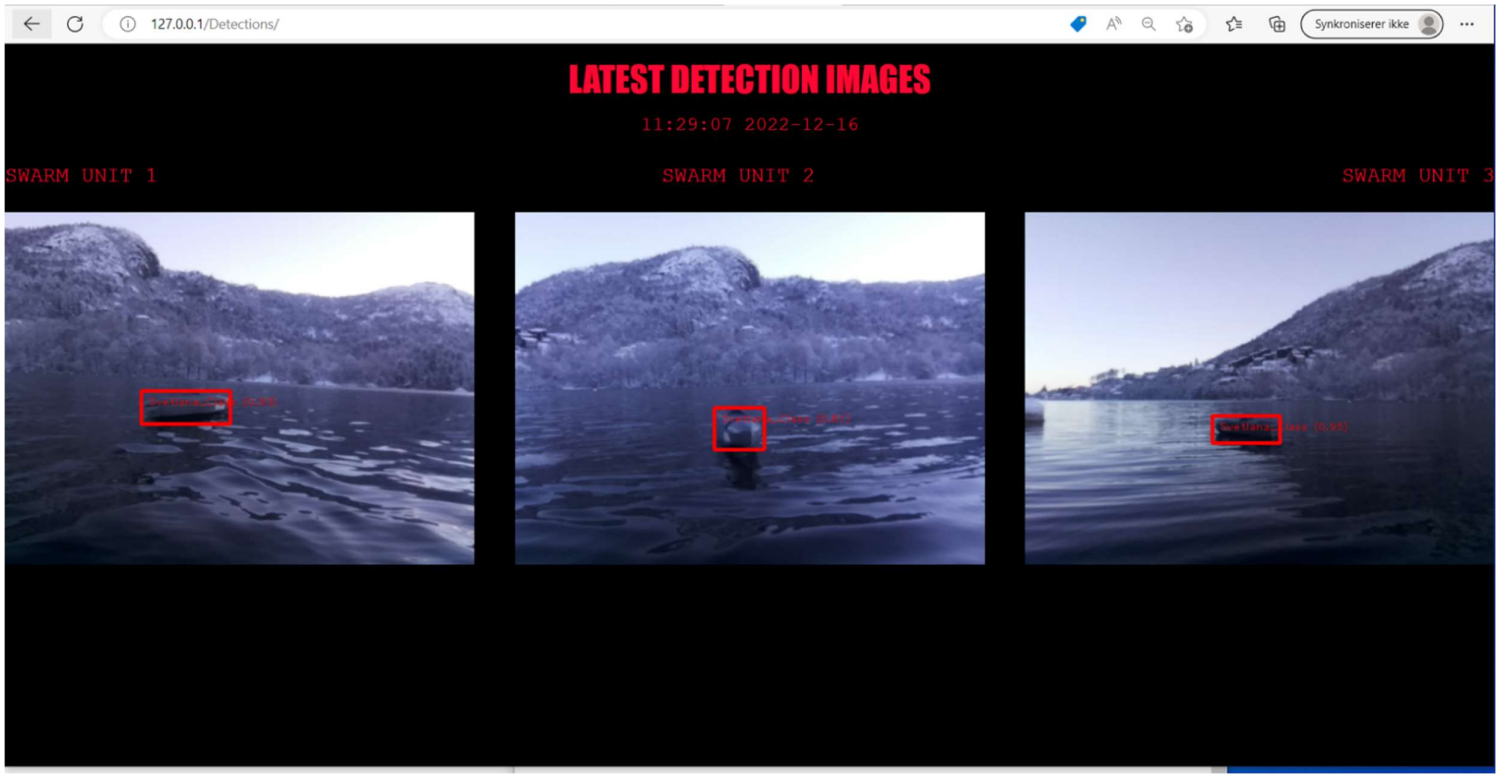
Figur 3-16: Utklipp av brukergrensesnittet fra sverm oppgaven fra 2019 (Hellesnes & Lyssand, 2019)

Videreutviklingen har fokusert på å vise deteksjonene til dronesvermen på en enkel og robust måte. Brukergrensesnittet skal kunne:

1. Vise deteksjonsbilder fra alle enhetene i svermen i tilnærmet nåtid.
2. Vise absolutt peiling av deteksjoner på kartet i forhold til enhetens daværende posisjon, basert på den relative peilingen som regnes ut på svermenhetene.
3. Muliggjøre krysspeiling av posisjonen til en deteksjon om flere enheter i svermen detekterte samme fartøy.

Første punkt ble løst ved å legge til en ny nettside som laster inn de nyeste deteksjonsbildene fra hver enkelt svermenhet dersom Svetlana er detektert over 80%. Hver svermenhet fungerer som en webserver og lagrer de nyeste deteksjonsbildene som /var/www/html/Detection_img_png. Nettsiden henter bildene fra webserveren på hver enhet og samler dem på nettsiden. I vårt tilskudd til brukergrensesnittet er det tilrettelagt for tre droner. Bildene blir hentet hvert 10 sekund. Bildet av siste deteksjon blir vist helt

til neste deteksjon av Svetlana over 80% blir gjort av en enhet. Dette operatørpanelet kalles HMI_2 og er vist i figur 3-17.



Figur 3-17: HMI_2 med deteksjonsbilder fra tre svermenheter

Tilskuddet til brukergrensesnittet tilføyer en ny dimensjon for operatør av svermen. Ved å ha mulighet til å se hva hver enkelt drone ser, kan operatør bedre sin situasjonsforståelse og dermed bedre sitt eget beslutningsgrunnlag.

Andre punktet vi ønsket å tilføye var muligheten til regne ut absolutt peiling på deteksjoner. Dette har blitt gjort ved å legge til funksjoner som kobler posisjon- og bevegelsesdata fra hver enhet med deteksjonsdataen, som nå er lagt inn i svermkommunikasjonen. For å holde oversikt over de ulike brukergrensesnittene har dette brukergrensesnittet fått navnet HMI_1. For å vise peilingen i HMI_1 operatørpanelet la vi til en mottaks- og reformateringsnode som tar imot deteksjonsdataen: header.id, relative_bearing og right_detection. Og ved en right_detection = true blir relativ peiling og deteksjons-id paret med posisjonsdataen til svermenheten med tilsvarende id. Dette blir vist i figur 3-18. Videre presenteres det på kartet som en rød linje med utgangspunkt i posisjonen svermenheten hadde på deteksjonstidspunktet.

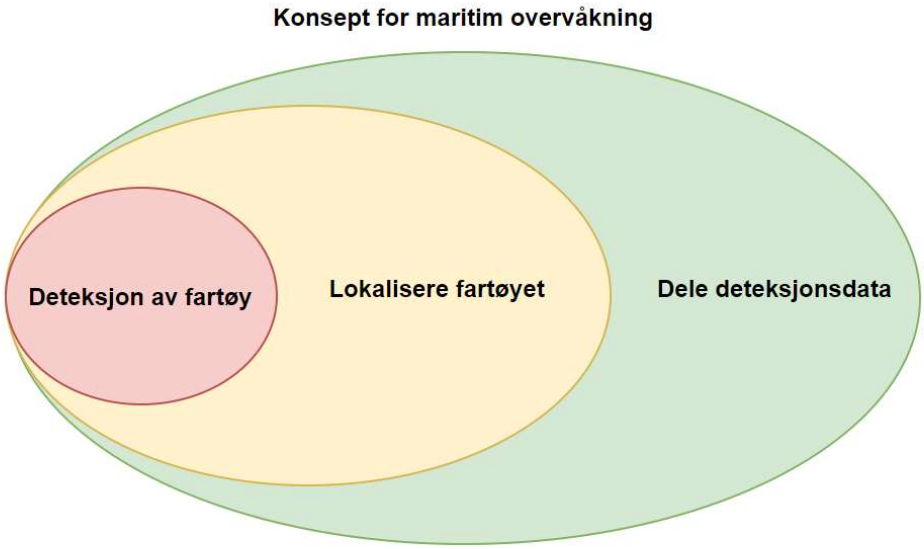
```
var Detection_for_map = msg.payload;
if (Detection_for_map.detection.right) {
    msg.payload.layer = "Detection";
    msg.payload.name = "Detection " + Detection_for_map.position.id;
    msg.payload.id = Detection_for_map.position.id;
    msg.payload.time = Date.now();

    //Regner ut absolutt peiling basert på båtens peiling fra GPS og relativ peiling fra deteksjon
    msg.payload.bearing = Detection_for_map.position.bearing + Detection_for_map.detection.rel_bear;

    //Plasserer peilingen på båtens daværende posisjon
    msg.payload.lat = Detection_for_map.position.lat;
    msg.payload.lon = Detection_for_map.position.lon;

    //Utseende til peilingen
    msg.payload.icon = "arrow";
    msg.payload.iconColor = "red";
    msg.payload.speed = 6;
    node.send(msg);
}
```

Figur 3-18: Kildekode fra node-red

Muligheten til å vise absolutt peiling, gjør det også mulig å kunne sette absolutt posisjon i kartet. Dersom to droner har peilet det samme fartøyet vil krysningspunktet mellom de røde deteksjonslinjene fra hver drone være posisjonen til fartøyet man ønsker å lokalisere. På denne måten kunne det vært mulig å finne posisjonen på interessante mål. Likevel ville dette vært en manuell prosess for operatør, som selv måtte satt posisjonen i kartet.

4 Tester og resultater

Oppgaven har til nå tatt for seg bakgrunnen for oppgaven, teorien for å forstå teknologien bak konseptet samt hvordan teknologien har blitt implementert. Videre skal vi nå teste konseptet. For at konseptet skal fungere må målene satt i tabell 1 nås. Disse målene bygger på hverandre og må fungere sammen for å gi brukbar data for maritim overvåkning. Figur 4-1 viser hvordan de ulike programdelene er avhengig av hverandre. For at dronene skal kunne lokalisere fartøy, må deteksjonene være nøyaktige i innboksingen av fartøyene og gi prediksjoner over 80% på Svetlana-klassen. For at systemet skal kunne dele aktuell deteksjonsdata med svermen og omverdenen, må relativ peiling kunnen regnes ut uten store feil. Av den grunn starter kapittelet med deteksjon av fartøy for og så gå inn på lokalisering og deling av deteksjonsdata. I siste delen testes også lagring av data.



Figur 4-1: Konseptets grunnoppbygning

I deteksjonsdelen vil ulike versjoner av *Svetlana_Detector* bli testet ut for å finne ut av hvilken av modellene som er mest hensiktsmessig. I denne delen vil testdata bli samlet fra en RPi3B+ og en RPi4 for modellene for å skaffe en forståelse for hvordan prestasjonen til modellene er på ulik maskinvare. Neste del av testene innbefatter test av relativ peiling, sending av data, implementering i svermkommunikasjon og til slutt absolutt posisjon. Den siste delen av kapittelet vil ta for seg test av datalagring.

4.1 Deteksjon av fartøy

I målene for deteksjon av fartøy, fra tabell 1, skal den videretrente modellen kunne gjenkjenne Svetlana-klassen med en sannsynlighet på over 80%. I denne delen av oppgaven vil vi vise til resultater fra testdatasettet samt testing på vannet. Videretrening på eget datasett vil også bli testet i denne delen av oppgaven.

4.1.1 Resultater fra testdatasettet

Denne testen søker å gi forståelse for hvor god den videretrente modellen er på å gjenkjenne Svetlana. Testen benytter seg av standard målenheter for «MS-COCO»-datasettet (Lin, et al., 2014). Første måleenhet for denne testen er «mAP Svetlana Class» og beskriver gjennomsnittlig presisjon på gjenkjenning av Svetlana. Mer presist innebefatter dette hvor mange korrekte prediksjoner som har blitt gjort av Svetlana på testsettet. «mAP Swarm Unit» måler samme parameterne for Swarm Units. Den andre måleenheten som blir sentral i testen er «AR». Denne viser hvor mange rette prediksjoner som er gjort, i forhold til hvor mange rette prediksjoner som var mulig.

Fremgangsmåten for testen er å videretrent modell opp mot et testdatasett. Testdatasettet består av 44 bilder med ulik avstand og vinkling på objektet. Viktig å presisere at modellen aldri har sett disse bildene før og gjør dermed testen til et godt estimat på hvor godt modellen vil fungere i møte med den virkelige verden. Testingen er gjort med kildekoden til TensorFlow og kildekoden kjøres i *Google Colabs Notebook* Abadi, et al., 2015).

De ulike modellenes påvirkning på ulik maskinvar vil også bli målt. Dette måles ved frames per second (FPS) og forsinkelse. FPS er en måleenhet på antall bilder som blir prosessert per sekund. *Detect.py* regner allerede ut FPS så det kunne derfor sendes til tekstfil for hver gang det blir regnet ut i scriptet. For denne måleenheten skal det hentes ut 10 verdier og regne ut et gjennomsnitt. Avslutningsvis vil også forsinkelse bli målt. Forsinkelsen måler tidsdifferensen mellom at en endring skjer og at det fanges opp av modellen. Graden av forsinkelse ble målt manuelt ved å hurtig dekke til kamera og observere når videostrømmen ble svart. Tiden ble tatt ved hjelp av stoppeklokke. Her er det en potensiell feilkilde i reaksjonstiden til mennesket.

Testene setter modellene Svetlana_Detector0 og Svetlana_Detector1 opp mot hverandre. Presisjonen til Svetlana_Detetor4 vil også bli testet for å sammeligne modellene mot et

større nettverk. Videre vil også FPS og forsinkelse testes på RPi3B+ og RPi4 opp mot de ulike modellene.

Resultatene fra testene vises i tabell 4

	Modeller		
	Svetlana_Detector0	Svetlana_Detector1	Svetlana_Detector4
mAP Svetlana Class	67.30%	76.76%	80.83%
mAP Swarm Unit	68.96%	71.73%	64.35%
AR	72.14%	76.90%	77.62%
FPS (RPi 4)	5.62 Hz	2.60 Hz	Ikke målt
FPS (RPi 3b+)	2.78 Hz	1.64 Hz	Ikke målt
Forsinkelse (RPi 4)	1.87 s	2.92 s	Ikke målt
Forsinkelse (RPi 3b+)	2.73 s	4.59 s	Ikke målt
Størrelse	4.34 MB	5.80 MB	20 MB

Tabell 4: Resultater fra testdatasettet

Resultatene presentert i tabell 4 viser differens på 9.36% i «mAP Svetlana Class» mellom Svetlana_Detector0 og Svetlana_Detector1. Dette tyder på en betydelig bedre presisjon på prediksjonene. Det kan også tydes at bedre presisjon fører med seg en nedgang i hastighet målt i FPS. Disse resultatene viser at svermenheten med RPi4 og Svetlana_Detector1 antagelig vil være bedre til å detektere Svetlana enn svermenheter med RPi3b+ med Svetlana_Detector0.

I resultatene kommer det også frem at Svetlana_Detector1 og Svetlana_Detector4 er bedre til å predikere Svetlana-klassen enn vennlige svermenheter. Grunnen til det kan være at en større andel av bildene i treningssettet inkluderer Svetlana klassen enn svermenheter. Resultatene fra test av FPS og forsinkelse viser at RPi4 kjører modellene med høyere FPS og lavere forsinkelse enn RPi 3b+. Men resultatene antyder også at jo større modellene er, jo mindre FPS og større blir forsinkelsen. Dette viser hvordan størrelse på objektdeteksjonsmodeller kan påvirke hastigheten i utførelsen av prediksjoner. Det viser

også hvordan kraftigere maskinvare muliggjør kjøringen av større og mer presise modeller. Videre understreker dette viktigheten av tilgjengelig prosessorkraft siden prosesseringen skal foregå om bord svermenhetene i dette prosjektet.

Det kommer også frem i resultatene at samtlige modeller oppnår et AR-resultat på over 70%. Dette tilsier at modellene ikke har veldig mange falske negativer, altså Svetlana klasser og svermenheter som ikke ble detektert. Noe som i utgangspunktet er positivt, men med tanke på at deteksjonsmodellen sannsynligvis får flere muligheter til å detektere et fartøy som kommer inn i synsfeltet sitt så kunne vi akseptert et dårligere AR-resultat.

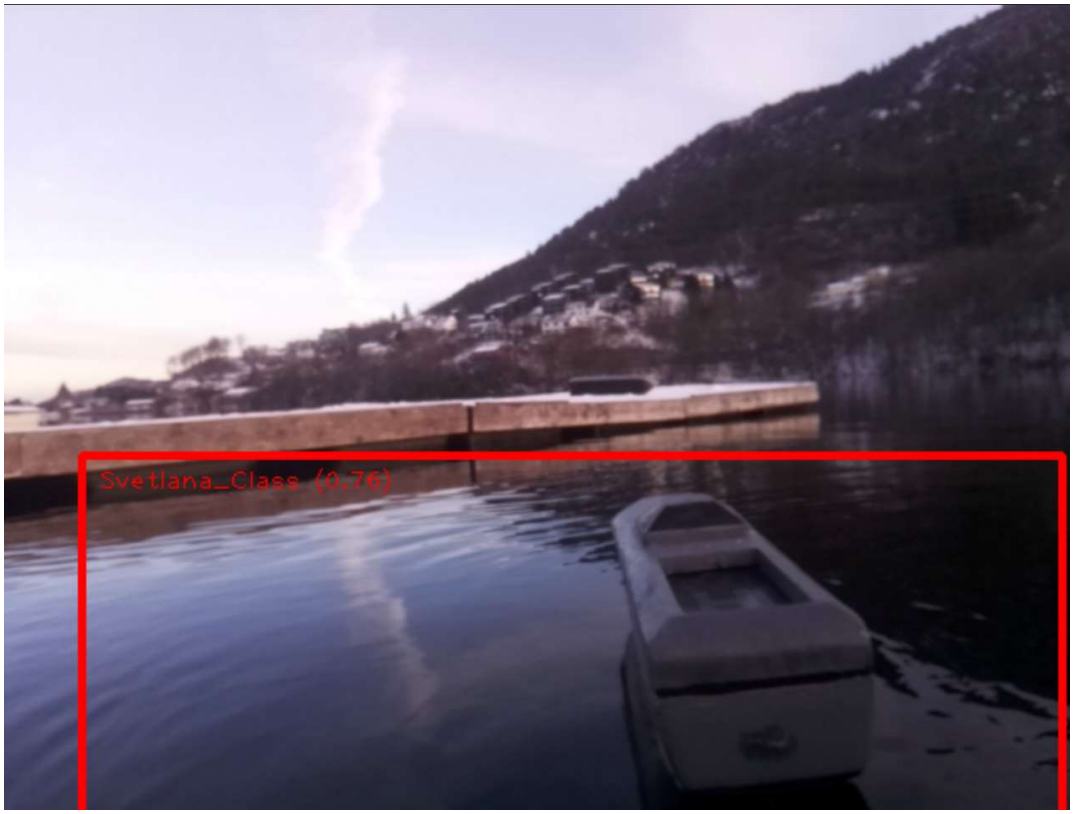
4.1.2 Første test på vannet

Etter presisjonen på videretrente modeller var målt det klar for første test på vannet. Testene skal få frem hvorvidt en videretrent modell, på få bilder, er tilstrekkelig for å gjenkjenne Svetlana-klassen. I denne testen benyttet vi oss av en drone med RPi4 som kjører Svetlana_Detector1. Testene skal også avdekke eventuelle sårbarheter for modellen. For å teste deteksjonsevnen og sårbarheten til modellen vil testene variere i avstand til- og vinkling på- Svetlana-klassen. Dette er fordi kompleksiteten øker med ulike distanser fra kameraet samt ulik vinkling på fartøyet som skal detekteres.

Første test var tiltenkt å gjøres på samme sted som treningsdatasettet ble tatt, men dette lot seg ikke gjøre. Vannet der bildene av Svetlana ble tatt var fryst til og gjorde at testene måtte bli gjort på et annet vann. Miljøet på vannet testene skulle foregå på, er distinkt forskjellig fra vannet modellen er trent på. Dette fører til at modellens sårbarheter kunne testes på en god måte. I tillegg til at miljøet rundt var endret, var også lysforholdene annerledes og gjorde at gjenskinnet i testvannet var tydelig. Det økte dermed kompleksiteten for å detektere Svetlana ytterligere for modellen. Flatt lys, annerledes miljø gjorde at modellen fikk flere konturer å holde styr på mens den skulle detektere Svetlana. Figur 4-2 viser forholdene under første test på vannet.

Resultatene for testene gjenspeiler at modellen ikke er trent for miljøet den ble testet i. Fra teststart ble det tydelig at forholdene rundt fartøyet har mye å si for hvorvidt modellen evner å detektere Svetlana korrekt. Modellen hadde problemer med å bokse inn båten på rett måte, slik man kan se i figur 4-2. I dette bildet ser man at boksen er altfor stor og

ikke tilpasset størrelsesforholdet til Svetlana. Dersom Svetlana hadde blitt detektert korrekt, ville boksen vært nærmest perfekt tilpasset størrelsen på skroget. Dette fører til manglende presisjon i utregning av relativ peiling senere i oppgaven. Likevel er det interessant at modellen likevel detektere klarer å gjøre deteksjoner på opp mot 80%, selv om fartøyet ikke er innbokset korrekt. Likevel er det plausibelt å anta at modellen ikke er tilpasset forholdene den ble testet på.



Figur 4-2: Skjermutklipp fra deteksjon av Svetlana under test 1

At modellen ikke er tilpasset testforholdene kan tydelig sees dersom treningsbildene blir sammenlignet med resultatbildene. Figur 4-3 viser et eksempelbilde fra datasettet modellen er videretrent på. Dersom bildene analyseres mot hverandre kommer det tydelig frem at lysforholdene er annerledes og at gjenskinnet under test er tydeligere enn under trening. Videre er også oppløsningen på bildene fra trening bedre enn resultatbildene. Dette er fordi modellen er trent på bilder som er tatt med et mobilkamera og ikke kameraet som skal benyttes. Av den grunn bestemmer vi oss for å videretrene modellen på nytt med bilder fra dronen, i de omgivelser som modellen skal testes på.

Figur 4-3: Eksempelbilde som modellen ble trent på før test 1

For videretreningen la vi til nye bilder i det eksisterende datasettet. Bildene ble tatt fra dronens perspektiv i det miljøet modellen skulle testes i. Et eksempelbilde fra datasettet etter første test kan sees i figur 4-4.

Figur 4-4: Eksempelbilde fra datasett til videretrening i nytt miljø

Testene fra videretreningen kommer frem i tabell 5. En interessant bemerkning fra videretreningen etter første test er forskjellen i mAP Svetlana Class mellom ny videretrent modell og gammel videretrent modell. Forskjellen viser at mAP Svetlana Class nå er lavere enn den tidligere har vært. Grunnen til dette ser vi flere muligheter til. Den ene er at modellen nå er trent for ulike miljøer og den blir utsatt for testbilder i flere miljøer. Et bredere spekter av testbilder vil kunne føre til et lavere mAP resultat. Den andre muligheten er at testsettet er større ved denne videretreningen og at mulighetene for å feile også øker. Uansett hva det skyldes er modellen nå trent på flere bilder og er klar for andre test på vannet.

Modell	Svetlana_Detector1
mAP Svetlana Class	76.52%
mAP Swarm Unit	74.43%
AP75	88.44%
AR	80.85%

Tabell 5: Resultater fra andre videretrening

4.1.3 Andre test på vannet

Etter ytterligere trening av modellen ble det gjort nye tester. Andre test ble gjort på samme sted som første test. Under andre test var igjen værforholdene endret. Det hadde kommet masse snø over natten og på dagen testene skulle gjennomføres. Dette vil være hensiktsmessig for testen av modellen, da den blir utsatt for et endret miljø. Følgelig vil testene vise hvorvidt modellen er mindre sårbar for endringer i miljøet etter videretreningen fra første test.

For å teste hvorvidt modellen er mindre sårbar, ble deteksjonsmodellen først testet med Svetlana på kort hold. Et eksempelbilde fra testen vises i figur 4-5. I eksempelbildet blir farkosten gjenkjent med en sannsynlighet på 95% og boksen er svært godt tilpasset størrelsen på skroget.

Figur 4-5: Skjermutklipp fra deteksjon av Svetlana under test 2, kort hold

Ut ifra testene på kloss hold antydes det at deteksjonsmodellen nå er mindre sensitiv på miljøet, og vi kan dermed øke kompleksiteten på testene. Neste test ble gjort med større avstand fra droneenheten. Avstanden ble målt opp med et fiskesnøre på 12 meter og Svetlana skyves ut til fiskesnøret er helt strukket ut. Et eksempelbilde fra testene vises i figur 4-6. Bildet viser at farkosten er klassifisert korrekt med en sannsynlighet på 82%. Videre er boksen rundt objektet også her godt tilpasset størrelsen på skroget.

Figur 4-6: Skjermutklipp fra deteksjon av Svetlana under test 2, langt hold

Neste test ble så gjennomført med ulike vinkler av Svetlana-klassen. I figur 4-7 er det et eksempelbilde fra testen. Dronen ser nå farkosten aktenfra på langt hold. Eksempelbildet viser at farkosten blir gjenkjent med 82% sannsynlighet. Bildet har også et lysforhold som gjør det vanskelig å tydelig skille ut Svetlana i bildet, fordi det er mørkt vann og landskap rundt Svetlana. I bildet er det verdt å merke seg at selv om avstanden til Svetlana har økt så er innboksingen rundt Svetlana fortsatt tett rundt skroget.

Figur 4-7: Skjermutklipp fra deteksjon av Svetlana under test 2, aktenfra

I denne testen ble det også avdekket noen sårbarheter for modellen. Når båten skulle plukkes opp, ble sannsynligheten målt på helt ned mot 50% selv om sannsynligheten før dronen ble plukket opp var på 75%. Eneste forskjellen på disse deteksjonene var en hånd som kom i bildet. Dette kan tyde på at modellen enda er sårbar for endringer i miljøet.

Videre kom det også frem at prediksjonssannsynligheten av Svetlana sank når deteksjonen var i kameraets ytterkanter, som også kan være en svakhet for modellen. Det ble også avdekket at modellen varierer veldig i sannsynlighetene for at det er Svetlana. I et spenn av 4 lagrede deteksjoner varierte sannsynlighetene mellom 51% og 84%. Dette kan tyde på at modellen ikke alltid gir jevne sannsynligheter, selv over korte tidsspenn.

Figur 4-8: Skjermutklipp falsk positiv deteksjon av Svetlana under test 2

Som vist I figur 4-8 så gjør modellen også sporadiske falske deteksjoner av Svetlana, selv uten fartøyer I synsfeltet. Selv om disse ofte har disse lavere sannsynlighet en de sanne positive deteksjonene av Svetlana så utgjør dette en feil med Svetlana_Detector modellen. Det er også plausibelt å anta at om modellen hadde vært utsatt for bilder som inneholder fartøyer som ligner Svetlana klassen, så kunne disse blitt feilaktig detektert.

Figur 4-9: Skjermutklipp av falsk negativ deteksjon av Svetlana under test 2

Resultatet som er vist i figure 4-9 at Svetlana_Detector1 modellen ikke alltid klarer å gjenkjenne Svetlana. I dette bildet ligger Svetlana med skutesiden mot kamera, som vanligvis er det som gir best sannsynlighet for deteksjon. Avstanden til Svetlana i dette bildet er også kortere enn på mange av bildene som har en sann positiv deteksjon. En detalj som kan gjøre det vanskelig for deteksjonsmodellen i dette bildet er at dekket på Svetlana ligger helt på linje med vannlinja. Dette kan gjøre det vanskelig å skille Svetlana fra landskapet bak.

Samlet sett så viser de fleste skjermutklippene fra andre test på vannet at Svetlana_Detector1 modellen har en betydelig bedret evne til å detektere Svetlana fra første test på vannet. Det er mange skjermutklipp med en sannsynlighet for deteksjon på over 80%. Modellen klarer å detektere Svetlana forfra, aktenfra og fra siden. Det er også verdt å merke seg at modellen gjenkjenner Svetlana på varierende avstander, fra 1 meter til ca. 12 meters hold.

Disse resultatene vitner om at andre videretrening har gjort Svetlana_Detector mer fleksibel på hvilket miljø den skal operere i. Forskjellen på skjermutklippene fra første test på vannet til den andre testen er tydelig. Sannsynligheten for deteksjon har økt og innbokseringer av Svetlana er blitt mye tettere på skroget. Og dette til tross for at mAP resultatet faktisk gikk ned etter andre videretrening.

4.2 Lokalisering av fartøy og deling av deteksjonsdata

Fra testene på deteksjon av fartøy kom det frem at modellen evner å detektere fartøy med en sannsynlighet over 80%. Videre går oppgaven over til å teste relativ peiling og dataoverføringen mellom RPi_ML og RPi-SV.

4.2.1 Relativ peiling

Relativ peiling er essensiell for å tilby nyttig deteksjonsdata til dronesvermen. Det er av den grunn viktig å avdekke eventuelle feilkilder og tilpasse systemet dersom feilen er tilstrekkelig stor. Av den grunn ble relativ peiling testet. Måleenhetene som har blitt be-

nyttet i denne testen er gjennomsnittlig avvik fra faktisk verdi samt standardavvik. Gjennomsnittlig avvik skal avdekke gjennomgående feil, mens standardsavviket skal avdekke hvorvidt resultatene fraviker fra hverandre i stor grad.

Oppsettet for testene innebefatter en RPi som kjører objektdeteksjon, med kameraet rettet mot tavlen. Kameraet ble satt vinkelrett på tavlen med en avstand på 2.985 meter. På tavlen ble det markert opp ulike grader som utregningen av relativ peiling skulle settes opp imot. Punktene ble det satt opp ulike punkter vi ønsket å teste på. Punktene ble regnet ut ved hjelp av vinkel i trekant. Dette ble gjort med følgende formel:

$$\text{Ønsket punkt på tavle} = 2.985 * \sin(\text{vinkel for test})$$

For å vise sammenhenger mellom data tok testene utgangspunkt i 5-, 10-, 15-, 20- og 30-grader. Dette forsøket ble gjennomført med den originale objektdeteksjonsmodellen, EfficientDet_Lite1. Denne modellen gjenkjenner generelle klasser, som for eksempel «person». Vi kunne derfor sette en person på hvert punkt vi ønsket å teste på. Oppsettet kan sees i figur 4-10.

Figur 4-10: Oppsett for test av relativ peiling

For hvert punkt ble det gjennomført 20 målinger. Alle målinger som ble gjort er lagt til i Vedlegg A. Ved oppstart av testene ble det oppdaget et problem med kalibreringen av kameraet. Det kommer frem fra figur 4-10, at synsvinkelen til kameraet er over 62.2 grader. Derfor ble kameraet kalibrert til synsvinkelen på tavlen. Dette gjorde at verdien for synsvinkel i *get_relative_bearing.py* måtte endres fra 62.2 grader til 75.1 grader.

Etter kalibreringen av kameraet startet testene. Ut ifra denne tabellen har vi regnet ut gjennomsnittlig målinger samt gjennomsnittlig feil over 20 målinger. Gjennomsnittlig feil har blitt regnet ut med følgende formel:

$$\text{Gjennomsnittlig feil} = (\text{Reell verdi}) - (\text{Gjennomsnittlig måling})$$

Ut ifra tabell 6 kan vi se at målingene har en gjennomgående feil i alle testene. I gjennomsnitt fraviker resultatene fra testene med en gjennomsnittlig verdi på -1,54 grader. Resultatene er ikke overraskende da det finnes lite i realiteten som er helt uten feil. Feilen kan skyldes unøyaktighet i kalibreringen av kameraet og unøyaktigheter i testoppsettet. Begge ble satt opp manuelt noe som kan være en feilkilde. En annen feilkilde kan være at innboksingen av personen i testen ikke treffer presist nok på personen. Et eksempel på dette kan sees i figur 4-10. Her trekker boksen noe mer mot negativ side og dette er også tilfellet i flere av bildene fra testen. Likevel er ikke dette ødeleggende for systemet.

	Grader				
	5	**10**	**15**	**20**	**30**
Gjennomsnittlig måling	6,0003	11,216	16,874	22,522	31,0989
Gjennomsnittlig feil	-1,0003	-1,216	-1,874	-2,522	-1,0989
Standardavvik	0,16278	0,17482	0,14209	0,23025	0,12640

Tabell 6: Resultater fra test av relativ peiling

Gjennomsnittlig feil gjennom alle testene er på -1,54 grader. Dette tilsvarer en feil på 13.12 piksler på modellen og en avstand på 8 centimeter ut ifra testoppsettet vårt. Feilen er dermed ikke så stor at den gir store utslag for systemets funksjon i helhet. Videre er standardavvikene fra hver av målingsgradene, over 20 målinger, heller ikke stor. Fra dette kan det tydes at utregningen av relativ peiling er relativt presis, men at systemet har feil som følge av kalibreringen av kameraet, oppsett av test eller en gjennomgående feil i innboksingen av objekter.

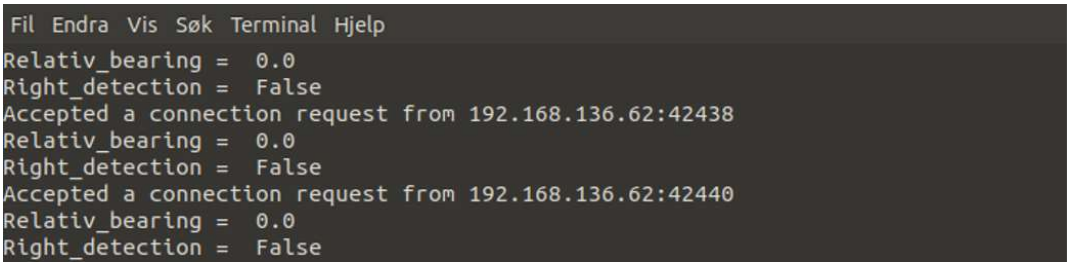
4.2.2 Dataoverføring mellom RPi_ML og RPi_SV

Testene av utregningen av relativ peiling er gjennomført og fører oppgaven over på test av deling av deteksjonsdata. Dataoverføringen mellom RPi_ML og RPi_SV på hver av droneenhetene skal sendes via en TCP-server. Denne delen av programmet er viktig for

at deteksjonsdata skal kunne implementeres i svermkommunikasjonen. Før deteksjonsdata kan implementeres inn i svermkommunikasjonen, må datakommunikasjonen mellom RPiene testes. Denne testen skal sjekke hvorvidt RPi_ML sender riktig data til RPi_SV. For å gjøre dette er det satt opp en server på RPi_SV som skal skrive innkommende deteksjonsdata. Videre startes hovedprogrammet, *detect.py*, på RPi_ML. Datakommunikasjonen skal testes i to ulike settinger. Settingene skal vise hvordan programmene reagerer med- og uten rett deteksjon.

Oppsettet og objektdeteksjonsmodellen som kjøres er den samme som i test av relativ peiling. I første setting skal kameraet stilles mot tavlen, uten objekter å detektere. I andre setting settes en person foran tavlen, slik at programmet får inn en deteksjon og regner ut relativ peiling og sender den. For testen har vi benyttet en RPi_ML med IP-adressen 192.168.135.62.

Ved første gjennomføring viser terminalvinduet at tilkoblingen fra rett IP er godkjent samt at det ikke er noen relativ peiling i denne situasjonen. Utklippet fra terminalvinduet vises i figur 4-11. Dette stemmer overens med settingen programmet er satt i. Av den grunn skal deteksjonsdata fra RPi_ML kunne implementeres i svermkommunikasjonen, dersom dataoverføringen også er tilstrekkelig i setting nummer to.



Figur 4-11: Teminalvinduet på RPi_SV for test av dataoverføring fra RPi_ML

I setting nummer to viser terminalvinduet en tilkobling til rett IP-adresse samt relativ peiling og at det er av en interessant deteksjon. Utklippet fra terminalvinduet vises i figur 4-12. Testene viste ingen avvik fra hva programmet skal gjøre. Resultatene fra testene viser at datakommunikasjonen fungerer og at deteksjonsdata dermed kan implementeres i svermkommunikasjonen.

```
Fil  Endra  Vis  Søk  Terminal  Hjelp
Relativ_bearing =  9.5
Right_detection =  True
Accepted a connection request from 192.168.136.62:42898
Relativ_bearing =  9.5
Right_detection =  True
Accepted a connection request from 192.168.136.62:42900
Relativ_bearing =  9.56
Right_detection =  True
```

Figur 4-12: Terminalvinduet på RPi_SV for test av dataoverføring fra RPi_ML

Et av hovedmålene i oppgaven var å dele deteksjonsdata med resten av svermen og operatørpanelene. For å gjøre dette må svermkommunikasjonen testes. I denne testen kjøres en RPi_ML med den originale modellen, som i test for relativ peiling samt test av dataoverføring. Objektet som skal detekteres er en person. Under testen kjøres RPi_ML med kameraet rettet mot personen. Samtidig kjøres RPi_SV som setter opp serveren RPi_ML skal sende deteksjonsdata over. Meldingene som sendes fra RPi_SV sjekkes på operatørpanelet. Suksess fra denne testen vil være dersom to meldinger sendes. En melding med posisjon og bevegelsesdata samt en melding med deteksjonsdata. Dette er fordi disse meldingene er av ulik meldingstype.

Figur 4-13 viser at RPi_SV sender to meldinger. Den ene meldingen inneholder informasjon om posisjon, fart og heading på linje 2. Dette kan sees på den øverste meldingen på linje to. Den andre meldingen viser relativ peiling og at det er gjort en deteksjon. Resultatet viser at deteksjonsdata er lagt inn i svermkommunikasjonen.

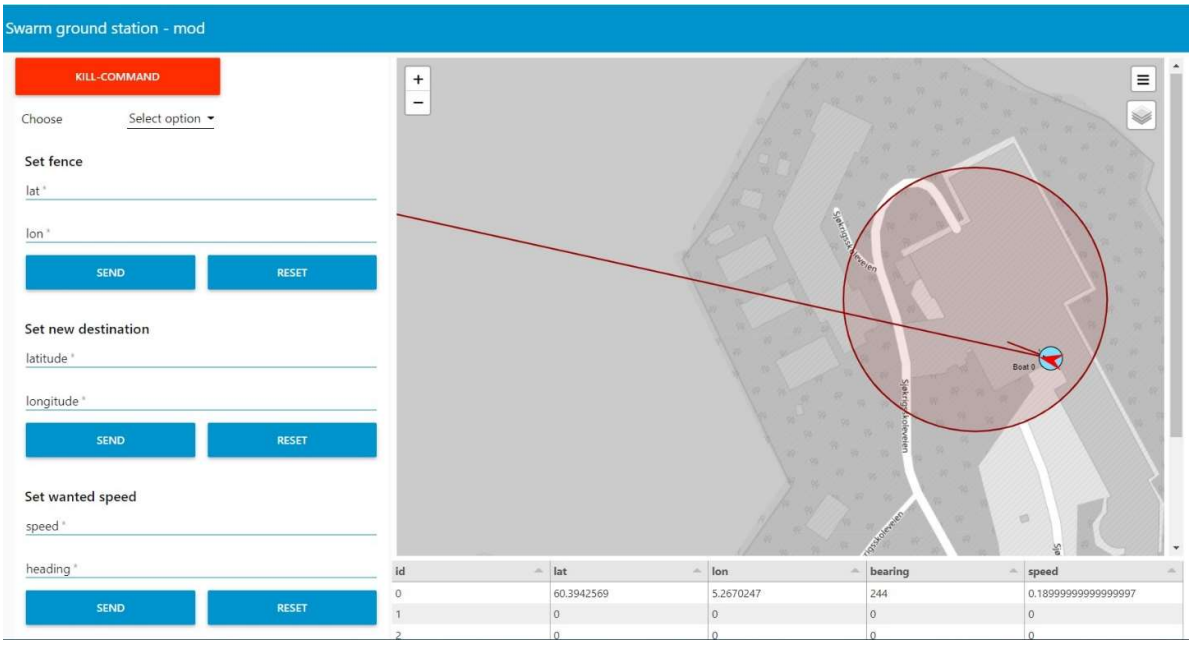
17.12.2022, 20:04:33 node: debug 1
msg.payload : string[220]
"{"header": {"reTransmit": 0, "seq": 0, "Ack": 0, "nsecs": 687854051, "secs": 1669381516, "msgType": 1, "id": 3},
"movement": {"speed": 2.35, "heading": 332.0}, "position": {"latitude": 60.396889, "longitude": 5.2663051}}"

17.12.2022, 20:04:37 node: debug 1
msg.payload : string[182]
"{"header": {"reTransmit": 0, "seq": 0, "Ack": 0, "nsecs": 687854051, "secs": 1669381516, "msgType": 4, "id": 3},
"Detection": {"relative_bearing": 2.393889, "right_detection": true}}"

Figur 4-13: Test av deteksjonsdata i svermkommunikasjon

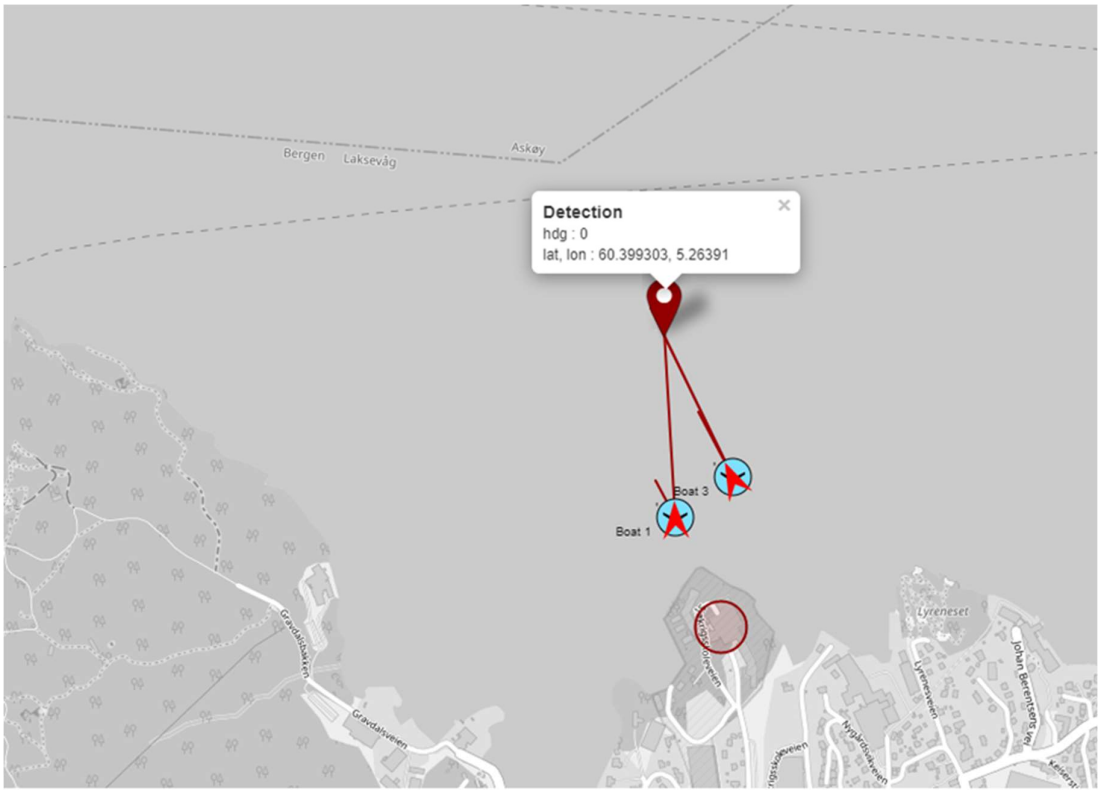
4.2.3 Brukergrensesnittet

Testene for brukergrensesnittet HMI_1 søker å finne ut av hvordan deteksjonsdata blir visualisert. Måten testene ble gjort på var ved å benytte en drone med GPS på land. For testen av brukergrensesnittet benyttes originalmodellen EfficientDet_Lite, som kan gjenkjenne en person. Videre ble det plassert en person i synsfeltet til dronen. Figur 4-14 viser et utklipp av brukergrensesnittet under testen. Utklippet viser at dronen klarer å regne ut absolutt peiling ved hjelp av dronens posisjon, dronens peiling og relativ peiling til deteksjonen. Under testingen ble det ikke funnet noen avvik fra dette resultatet.



Figur 4-14: Utklipp fra test av brukergrensesnittet

Etter testene av brukergrensesnittet, ble det gjennomført en simulering for å finne absolutt posisjon. I dette forsøket ble to simulerte droner satt ut på sjøen. Dronene fikk en satt posisjon og peiling. Videre ble denne informasjonen tilføyet av deteksjonsdata fra hver av de simulerte droneenhetene. Meldingene ble sendt på samme format som meldingene blir sendt i ROS, men via en simuleringsnode i Node-Red. Simuleringen vises i figur 4-15. Posisjonen etter krysspeiling som vises i brukergrensesnittet, måtte settes manuelt av bruker under testen. Programmet klarer ikke selv å beregne posisjonen. Simuleringen er gjort med simulerte droner, men resultatet tyder på at svermen kan tilby muligheten for å finne absolutt posisjon. Såfremt operatøren av operatørpanelet manuelt trykker på punktet de to absolutte peilingene, krysser.

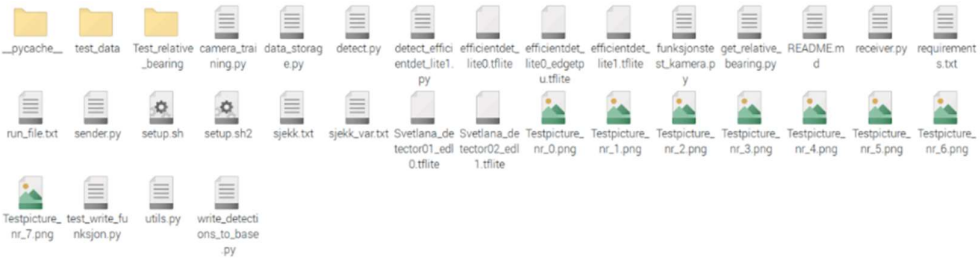


Figur 4-15: Simulering av absolutt posisjon

4.3 Lagring av data

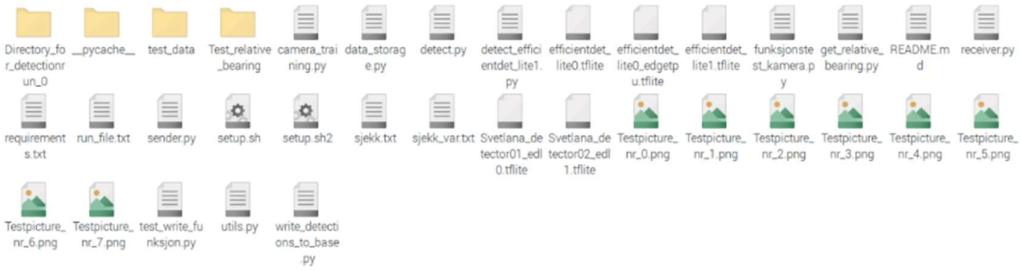
Lagring av data er en viktig del for videretreningen av modellen. Et av målene satt i tabell 1 er at hver enkelt drone skal kunne lagre data lokalt. Dette gjøres for å legge til rette for å utvikle modellen, selv etter svermen har blitt tatt i bruk. Av den grunn er det viktig at en mappe blir opprettet for hver gang programmet kjøres samt at bilder og log-data lagres underveis. Testen gjøres ved at programmet kjøres og resultatene sjekkes fysisk inne i mappestrukturen på RPi'en som kjører programvaren for testen. Modellen som blir kjørt er EfficientDet_Lite og det stilles en person foran tavlen.

Mappestrukturen ser slik ut før programmet kjøres, se figur 4-16. På denne RPien har ikke deteksjonsprogrammet blitt kjørt før og det er dermed ingen mapper for lagring av bilder. Det gjør det enklere å vise endringer i mappestrukturen.



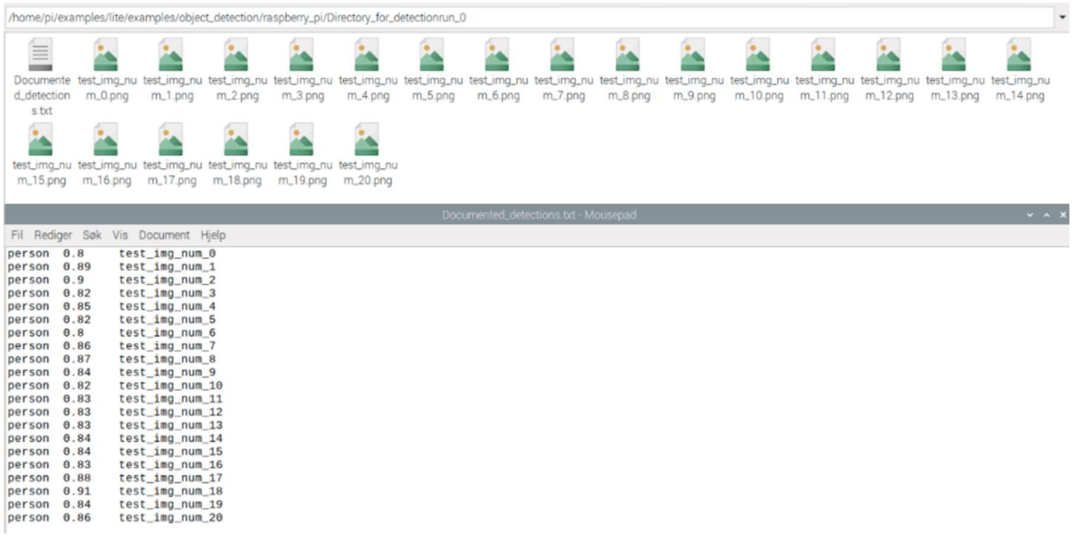
Figur 4-16: Mappestruktur før kjøring av program

Hovedprogrammet blir så kjørt og mappen «Directory_for_detectionrun_0» blir opprettet i mappestrukturen. Dette vises i figur 4-17.



Figur 4-17: Mappestruktur etter programstart

Mappestrukturen åpnes kort tid etter start av programmet. I mappen har det nå blitt lagt til 20 bilder av deteksjoner. Figur 4-18 viser dette samt at det har blitt lagt til en loggfil. I loggfilen er det lagret samme antallet bilder som er kommet inn og deteksjonen skriver rett klasse med tilhørende sannsynlighet sammen med bildet som er lagret.



Figur 4-18: Inne i mappe under kjøring av hovedprogrammet

5 Drøfting

I dette kapittelet skal resultatene fra testingen drøftes opp mot målene som har blitt satt i oppgavene. Videre skal prosjektet settes i en operasjonell kontekst for å diskutere potensialet dette kan ha i Sjøforsvaret.

5.1 Resultater drøftet mot mål for oppgaven

I denne oppgaven har det blitt utviklet et system for fartøysdeteksjon på dronesverm. Produktet skulle gi dronene i svermen en bedre situasjonsforståelse, ved å anvende objektdeteksjon. Dette har blitt gjort gjennom modifisering av et deteksjons-script som utnytter output fra objektdeteksjonsmodellen. Fokus gjennom hele prosjektet har vært å implementere dataen inn i svermkonseptet fra 2019. (Hellesnes & Lyssand, 2019)

Mål	Beskrivelse
Deteksjon/Identifikasjon	Kunne detektere egendefinert klasse med presisjon høyere eller lik 80% mAP
Lokalisering	Enhetene skal internt kunne regne ut relativ peiling ved hjelp av kamera og deteksjonsmodellen
Deling	Deteksjonsdata skal kunne kommuniseres til andre enheter i svermen samt operatørpanelet
Samler deteksjonsdata til videre trening	Enhetene skal kunne lagre bilder ved deteksjon for videre trening av deteksjonsalgoritme.

I oppgaven var målet for deteksjonsevne å kunne detektere egendefinert klasse med presisjon høyere eller lik 80% mAP. Resultatene etter første videretrening viser at Svetlana_Detector1 oppnådde et resultat på 74.25% mAP. Dette resultatet er relativt nære, men svarer ikke til målene vi hadde satt for deteksjonsevne. Resultatet kan skyldes flere faktorer. Den første faktoren er at treningsdatasettet ikke har nok bilder av Svetlana-klassen. Det er plausibelt å anta at modellen ville fått høyere mAP dersom dette hadde vært

tilfellet. Her kunne man muligens også fått høyere mAP ved å trene opp en modell fra bunnen av, ikke ved å benytte *transfer learning*. Men på en den andre siden så hadde det krevd et veldig mye større datasett av bilder med Svetlana klassen enn det som ble benyttet i prosjektet. Den andre faktoren er at Svetlana_Detector1 er basert på EfficientDetLite1, som er en lettvekts deteksjonsmodell laget for edgeenheter. Bakgrunnen for at vi valgte denne modellen var datakraften vi hadde tilgjengelig på RPi4.

RPi ble vårt valg av datamaskin. Dette er små datamaskiner med lav prosesseringskraft og kan ikke kjøre de store nevrale nettverkene, da det gir forsinkelser og lav FPS. Fra resultatene i tabell 4 ser vi at Svetlana_Detector4 oppnår et resultat på 80.83% mAP. Denne modellen er basert på EffictientDetLite4, som er et større nettverk. For å anvende et slikt nettverk kunne en kraftigere datamaskin blitt benyttet. En annen mulighet hadde vært en USB-akselerator. USB-akseleratorer kjører deteksjonsmodellen via en USB-port og ville tilført prosessorkraft til systemet. Likevel viser prosjektet vårt et viktig poeng. Selv om det ikke er den ideelle løsningen, viser den at komplisert teknologi kan anvendes på små datamaskiner og gi relativt gode resultater som vist i 4.1.3 Andre test på vannet.

Resultatene viser at Svetlana_Detector1 klarer å gjøre riktige prediksjoner i varierende situasjoner. Prediksjonene ble her faktisk ofte gjort med over 80% sannsynlighet for Svetlana. Eksempelbildene fra testene viser at modellen klarer å gjenkjenne med varierende avstander og i ulike vinklinger av Svetlana. For konseptet med maritim overvåkning vil det heller ikke være nødvendig med høy prediksjonssannsynlighet på alle deteksjoner. Det maritime domenet er preget av relativt lav hastighet og dermed kan man anta at droneenhetene vil få flere muligheter til å detektere over 80% om noe først entrer synsfeltet – særlig for sverm med mange enheter. Man kan likevel argumentere for at det er viktigere å unngå *false positives*, enn at de fleste deteksjonene gjøres over 80%, men dette er avhengig av konteksten rundt systemet.

Et av eksempelbildene fra 4.1.2 viser er at det oppstår falske positiver under testene. Det vil si at modellen gjenkjenner Svetlana selv om den ikke er i bildet. Dette kan oppstå fordi det finnes linjer i bildet som minner om de linjer modellen har lært seg å gjenkjenne Svetlana med. Her kunne konseptet med dronesverm kommet til rette, en sverm av sjødroner vil kunne få flere sensorer på potensielle deteksjoner og tilføyer dermed muligheten for å bekrefte eller avkrefte deteksjoner. Et annet problem som oppstår i testbildene er at

Svetlana ikke alltid detekteres altså falske negative resultat. Dette kan komme av at modellen vår ikke er videretrent på en tilstrekkelig mengde bilder. Disse to feilkildene gjør at systemet kan ha bruk for en operatør som overvåker og kan verifisere deteksjonsbildene som dukker opp i operatørpanelet. Det kan likevel argumenteres for at en operatør av systemet er ønskelig uansett. Et system for maritim overvåkning skal tilføye situasjonsforståelse og det er derfor uviktig om det tidvis blir sendt inn tomme bilder. Det som er viktig er at operatør har muligheten til å motta deteksjoner på interessante fartøy og dermed øker situasjonsforståelsen i området svermen opererer i.

Et annet viktig poeng er hvor avhengig modellen er av miljøet den er trent på. Dersom modellen opererer i et miljø den ikke er trent på, vil resultatene bli dårligere. Dette kommer frem i resultatene fra 4.1.2, der modellen ikke klarer å detektere Svetlana. Lysforhold og omgivelser som modellen er trent på påvirker resultatene i stor grad. Dette er noe som er svært viktig å ta i betraktning dersom konseptet fra denne oppgaven skal tas i bruk. Ut ifra våre resultater er det plausibelt å anta at en modell ikke kan trenes for et miljø og deretter fungere optimalt i alle miljøer. For at konseptet skal fungere innenfor maritim overvåkning må deteksjonsmodellen trenes i mange miljøer, slik at den er mindre sårbar for endringer og dermed mer robust. Dette kom frem i bildene av andre testing på vannet 4.1.3, der modellen var trent på forskjellige miljøer og leverte merkbart bedre prediksjoner.

At modellene er miljøavhengige var en antagelse vi gjorde i starten prosjektet og har derfor lagt til en lagringsfunksjon i programvaren. Funksjonen tilrettelegger for kontinuerlig utvikling av objektdeteksjonsmodellen. Ettersom at lagringen foregår lokalt på hver enkelt enhet i svermen akkumuleres det et stort datasett totalt i svermen. Dermed vil objektdeteksjonsmodellen til svermen raskt kunne tilpasse seg miljøet den operer i.

Dette fordrer dog at treningssettet blir merket og at et godt system for å hente ut dataen er på plass. Bildene som er lagret på hver enhet må markeres og merkes med rett klassifikasjon. Dette er en tidkrevende prosess som krever menneskelige ressurser. Videre vil datamengden etter hvert bli stor og vil fordre en database med stor lagringsplass. Databasen må da tilrettelegge for å gjøre datasettet mest mulig oversiktlig.

Likevel kreves det at modellen er trent på fartøy av interesse for at systemet skal kunne gjenkjenne dem. Dersom datasettet ikke inneholder de klassene man ønsker å overvåke vil ikke disse gjenkjennes eller bli gjenkjent som noe annet. Dette kan være et problem

da interessante fartøyene sannsynligvis ikke oppholder seg i miljøet svermen skal operere i. Av den grunn mener vi at datasettet bør inneholde de fartøyene man ønsker å overvåke i alle relevante miljøer for å gjøre modellen mest mulig robust.

Et annet mål var at enhetene internt kunne regne ut relativ peiling ved hjelp av kamera og deteksjonsmodellen. Relativ peiling tilføyer situasjonsforståelsen til den enkelte drone og svermen som helhet. Likevel vil feil i utregning av denne føre til unøyaktig situasjonsforståelse. I våre tester klarer systemet å regne ut relativ peiling med en gjennomsnittlig feil på -1.54 grader. Dersom en deteksjon er en kilometer fra kameraet, vil dette føre til en posisjoneringsfeil på 26.9 meter. En lokaliseringsfeil i dette størrelsesområde være ikke ødeleggende for konseptet. Feilen kan også være mulig å utbedre, dersom systemet kalibreres nøyaktig i forhold til kameraets synsfelt.

Dette fører oss også videre inn på plasseringen av kameraet på dronene. Under testene av relativ peiling ble det tydelig at små feil på vinkling av kameraet vil ha stor innvirkning på hvor presis lokaliseringen er. Det er derfor svært viktig at kameraene er plassert nøyaktig i dronens kjøreretning og at kameraet er kalibrert. En svakhet i vår oppgave er at kameraholderne ikke er optimale. Kameraholderne er noe ustabile og det er vanskelig å orientere kameraet i dronens kjøreretning. Likevel ser vi at kameraholderne er tilstrekkelige for å kunne bevise konseptet i oppgaven.

Siste målet for oppgaven var at deteksjonsdata skal kommuniseres til andre enheter i svermen samt operatørpanelet. Dette ble løst ved å sende relativ peiling sammen med en boolsk variabel. Den boolske variabelen blir lik true dersom Svetlana detekteres og variablene sendes sammen med relativ peiling på deteksjonen. Den boolske variabelen ble lagt inn for å kunne bygge videre på svermkonseptet. Boolske variabler brukes i løsningen av diskrete problemer. Et diskret problem kan være hvilken atferd svermen skal ha i avhengighet til deteksjonen. Dersom det skal utvikles en svermalgoritme som agerer på deteksjonsdataen, vil det nå være enkelt å kunne utnytte den, men den boolske deteksjonsvariabelen kan skape en utfordring. Om deteksjonssannsynligheten går fra over 80% til under 80%, vil deteksjons variabelen bli satt fra true til false. Om sannsynligheten flimrer opp og ned i dette område kan det føre til at svermen skifter for ofte mellom atferder. Med for eksempel en timer som starter når deteksjonen går fra true til false, vil skiftene mellom atferder gjøres mer robust for maritim overvåkning, da svermen ville vært mindre avhengig av konstante verdier på deteksjonsdata.

Tidlig i prosjektet forsøkte vi å legge fartøysdeteksjonen på samme RPi som kjører svermen. Dette lot seg ikke gjøre uten problemer. Deteksjonsmodellen vi har benyttet oss av er avhengig av pakker for å kjøre, men operativsystemet på RPi_SV støttet ikke rett utgave av pakkene. Dette førte til at vi brukte mye tid fånyttes, uten at vi fikk tilpasset koden. Det var da vi utforsket muligheten for å benytte oss av en ny RPi om bord på hver av dronene. Etter å ha utforsket muligheten kom vi frem til at dette ikke bare var en løsning, men trolig også den beste løsningen. Med den innledende løsningen ville to ulike og krevende operasjoner blitt kjørt samtidig på samme enhet. For en liten PC som RPi ville dette ført til at mindre prosessorkraft gikk til svermstyring og følgelig det samme for fartøysdeteksjonen. Med den nye løsningen blir problemet løst, og programmene kan kjøres fint separat.

En svakhet i vår oppgave er antall piksler som blir benyttet i programmet (*detect.py*) som kjører modellen. Scriptet tar inn og prosesserer 640 piksler totalt i horisontal retning. Det vil si at dersom et 1X1 meter stort objekt plasseres 545 meter unna kameraet ville objektet oppfattes som en enkelt piksel i horisontal retning. Dette gjør programmet mindre hensiktsmessig for overvåkning over større avstander. Dette kunne blitt løst dersom mer en mer krevende modell hadde blitt tatt i bruk sammen med et bedre kamera. Dette hadde krevd større prosesseringskraft og ville vært vanskelig å få til på en RPi. Likevel ville datamaskiner med bedre prosesseringskraft kunne benyttet mer komplekse nettverk og dermed løst problematikken. I vår oppgave ble likevel programmene og modellen ansett som tilstrekkelig for å bevise konseptet i en svermkontekst. Dette er fordi flere droner vil kunne dekke et stort område totalt, men hver enkelt svermeenhet vil dekke et mindre område alene.

5.2 Potensialet i en sjømilitær kontekst

Prosjektet som er utført gjennom denne bacheloroppgaven kan tilføre en styrke innenfor maritim overvåkning gjennom evnen til å detektere og peile fartøy autonomt. Gitt at modellen er tilpasset miljøet og trent på rette fartøy vil svermen kunne operere alene. Det er også en fordel at personell ikke bindes opp i å drive bildebyggingsprosessen på lavest mulig nivå, og minimerer risikoen for tap av menneskeliv samt kritisk materiell. Likevel krever dette at infrastrukturen rundt svermen er tilrettelagt. Per nå kjøres svermen på batteri som med relativt hyppig frekvens må lades. Hvis Sjøforsvaret skulle hatt elektriske

sjødroner, ville dette krevd ladestasjoner eller autonome ladefartøy. Antagelig ville dette bundet opp folk dersom svermen ikke er automatisert for slike prosesser. Av den grunn kan det argumenteres for at potensialet for maritim overvåkning på maritime sjødronesvermer ikke kan utnyttes før infrastrukturen er på plass.

Maritime svermenheter med optiske sensorer som benytter objektgjenkjenning til å detektere overflatefartøy i et bestemt område har potensial til å komplementere Sjøforsvarets styrkestruktur. Svermteknologi og sjødroner kan benyttes til å dekke et større havområde ved hjelp av kvantitet. En sjødrone alene har ikke et veldig stort synsfelt, men eksempelvis 50 sjødroner som beveger seg rundt i et bestemt område har et stort synsfelt. Om man belyser dette synsfeltet med enkle optiske sensorer, og en objektgjenkjenningsmodell, så vil det være vanskelig å seile inn dette området uten å bli detektert. Kvantiteten gjør det også vanskelig for en potensiell aggressor å nøytralisere nok sjødroner til at en selv ikke blir detektert. En sverm med denne evnen kan for eksempel passe bra til å dekke havneområder og innløp i fjorder, områder som tilbyr dekke fra radar og der fartøy kan gjemme seg bak fjell, holmer og skjær.

En annen motsetning til bruk av konseptet vi har utviklet er at autonome systemer enda ikke er ideelle for oppgaven de skal gjøre. I en militær operasjon kan konsekvensene være store dersom beslutninger tas på feil situasjonsforståelse. Beslutninger som i verstefall kan ende med tap av menneskeliv. Av den grunn kreves det stor grad av pålitelighet fra systemene man skal ta beslutninger på bakgrunn av.

Det kan dermed diskuteres hvorvidt svermkonseptet for maritim overvåkning er tilstrekkelig godt. Testresultatene i denne oppgaven tyder på at svermenhetene klarer å gjøre gode deteksjoner på Svetlana. Dette var et av målene for oppgaven og viser at enkeltenhetene kan være tilstrekkelige for å bygge bilde. Videre kan det da tenkes at en sverm av disse enhetene sammen kan gi et enda bedre bilde av området det overvåker. Likevel er systemet langt fra perfekt og det kan tenkes at konseptet fra denne oppgaven heller bør benyttes som et verktøy for en operatør. Dette fordi resultatene i oppgaven også viser at det er flere falske positivt gjenkjente Svetlana. En falsk positiv deteksjon av et fiendtlig fartøy kan være farlig, fordi det potensielt skal leveres våpen på.

6 Konklusjon med anbefaling

I denne oppgaven har vi utviklet et system for deteksjon og identifikasjon ved hjelp av dronesverm. Systemet oppnår tilstrekkelig måloppnåelse for prosjektet, opp mot de kravene som ble satt. Av den grunn kan det sies å være et proof of consept. Programmene klarer å detektere egendefinert klasse, lagrer bilder av deteksjoner for videre trening, regner ut relativ peiling på detektert mål og sender relevant data til resten av svermen samt basestasjonen.

Resultatene i testene antyder god måloppnåelse opp mot satte mål og svarer dermed godt til intensjonen bak prosjektet. Det kunne blitt benyttet kraftigere datamaskiner for å kjøre en enda større og mer kompleks objektdeteksjonsmodell. Dette kunne gitt mer presise deteksjoner. Likevel er ikke dette essensen i oppgaven. Oppgavens intensjon var å vise hvor enkelt avansert objektdeteksjonsteknologi kan implementeres med begrensende ressurser.

Systemet er utviklet og testet for å bevise et konsept for bruk av objektdeteksjon for maritim overvåkning. Under prosjektutviklingen har fokuset vært tilrettelegging for videre utvikling av svermkonseptet. Slik som systemet er utviklet er deteksjonsmodulen av prosjektet involvert i svermkommunikasjonen og dermed klar for utvikling og testing med svermalgoritmer tilpasset søk og deteksjon av fartøy.

Videre kommer det frem fra resultatene i testene at deteksjonsmodellen er avhengig av miljøet. Dersom konseptet skulle bli tatt i bruk, kan det antydes at deteksjonsmodellen måtte blitt trent i flest mulig miljøer for å gjøre den mindre sårbar for endringer. Dette kan gjøres ved å samle data fra alle miljøene modellen skal operere i og lage et samlet datasett basert på det. Programmene for lagring av bilder muliggjør innsamlingen av bilder fra alle svermene som benytter seg av programmene i oppgaven og legger dermed grunnlaget for å danne et slikt datasett. Dette datasettet mener vi bør inneholde et base-datasett av fartøy som er av interesse. Vår anbefaling for base-datasettet er at det bør inneholde stor kvantitet for hvert enkelt fartøy man vil detektere.

Dette fører oss inn i anbefalinger for videre arbeid. Første anbefaling for videre arbeid er videreutvikling av svermalgoritme for maritim overvåkning. Underveis i prosjektet har det blitt utviklet en HMI som tilrettelegger for krysspeiling fra flere droner og en algoritme som utnytter dette til målfølging ville vært av interesse.

Andre anbefaling er å videreutvikle kommunikasjonen i dronesvermen. Per nå drives den på WiFi og er dermed ikke robust mot jamming.

Tredje anbefaling er å benytte deteksjonsmodellen til å drive kollisjonsunnvikelse for å bedre svermens miljøoppfatning ytterligere.

Fjerde anbefaling er å begynne prosessen med å hente inn bilder til datasettene allerede nå. Dette gjelder bilder av interessante fartøy og bilder fra ulike miljøer langs norskekysten. For å tilrettelegge for et moderne sjøforsvar kan dette være føre-var.

Videre har vi stor tro på konseptet og anbefaler videre arbeid med en litteraturstudie om hvordan autonome dronesvermer kan benyttes i Sjøforsvaret for maritim overvåkning. En slik studie kan legge til rette for å ta teknologien i bruk.

7 Bibliografi

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., . . . Jozefo. (2015). *TensorFlow: Large-scale machine learning on heterogeneous systems.* Google Research. Hentet fra tensorflow.org

Achmad, H., Priyandoko, G., Roali, R., & Daud, M. R. (2017). Tele-Operated Mobile Robot for 3D Visual Inspection Utilizing Distributed Operating System Platform. *International Journal of Vehicle Structures and Systems*. Hentet fra https://www.researchgate.net/figure/Nodes-communication-model-in-the-ROS-environment-Fig-5-describes-the-proposed-ROS_fig2_319566597

Allan, A. (2019). *Benchmarking TensorFlow Lite on the New Raspberry Pi 4, Model B.* Hentet fra Webområde for Hackster.io: https://www.hackster.io/news/benchmarking-tensorflow-lite-on-the-new-raspberry-pi-4-model-b-3fd859d05b98

Bozinovski, S. (2019, Juni 10). *Reminder of the First Paper on Transfer Learning in Neural Networks.* Hentet fra Informatica: https://www.informatica.si/index.php/informatica/article/view/2828

Challenger Aerospace Systems. (2019). *UNNMANNED SURFACE VEHICLE (USV) [bilde]*. Hentet fra https://challengeraerospace.com/innovations/unmanned-surface-vehicles-usv/

Chen, J. (2022, September 21). *What Is a Neural Network?* Hentet fra Investopedia: https://www.investopedia.com/terms/n/neuralnetwork.asp#:~:text=A%20neural%20network%20is%20a,organic%20or%20artificial%20in%20nature.

Chen, J., & Ran, X. (2019, August 8). Deep Learning With Edge Computing: A Review. *Proceedings of the IEEE*, ss. 1655-1671.

Egil, N. A. (2001). *Maskinlæringsteknikker for klassifisering.* Kjeller: FFI.

Elfa Distrelec. (2022). *PI CAMERA MODULE V2.1 [Bilde]*. Hentet fra Elfa Distrelec: https://www.elfadistrelec.no/en/raspberry-pi-camera-v2-raspberry-pi-pi-camera-module-v2/p/30134462?queryFromSuggest=true

Elfa Distrelec. (2022). *RASPBERRY PI B+ [Bilde]*. Hentet fra https://www.elfadistrelec.no/en/raspberry-pi-model-512mb-ram-raspberry-pi-raspberry-pi/p/30001887

Fairchild, C., & Harman, T. (2016). Getting Started with ROS. I C. Fairchild, & T. L. Harman, *ROS Robotics By Example: Bring life to your robot using ROS robotic application* (ss. 1-32). Packt. Hentet fra https://books.google.no/books?id=skFPDwAAQBAJ&printsec=frontcover&dq=ROS+Robotics+By+Example:+Learning+to+Control+Wheeled,+Limbed,+and+Flying+Robots+Using+ROS+Kinetic+Kame+Thomas+Harman&hl=en&sa=X&redir_esc=y#v=onepage&q=ROS%20Robotics%20By%20Example%3A%2

Hareide, O. S., Relling, T., Pettersen, A., Sauter, A., & Mjelde, F. V. (2018). Fremtidens autonome ubemannede kapasiteter i Sjøforsvaret. *Necesse*, ss. 123-148.

Helleland, L. H., & Astrup, N. (u.d.). *regjeringen.no.* Hentet fra Kyststrategi: https://www.regjeringen.no/no/dokumenter/kyststrategi/id2862477/?ch=1

Hellesnes, A. H., & Lyssand, K. (2019). *Plattform for sverm.* Bacheloroppgave, FHS Sjøkrigsskolen, Bergen, Norge.

IBM Cloud Education. (2020, Oktober 20). *Convolutional Neural Networks [Bilde]*. Hentet fra IBM: https://www.ibm.com/cloud/learn/convolutional-neural-networks

IBM Cloud Education. (2020, August 17). *Neural Networks*. Hentet fra IBM: https://www.ibm.com/cloud/learn/neural-networks

IBM Cloud Education. (2020, August 19). *Supervised Learning*. Hentet fra IBM: https://www.ibm.com/cloud/learn/supervised-learning#:~:text=Supervised%20learning%2C%20also%20known%20as,data%20or%20predict%20outcomes%20accurately.

Jeffares, A. (2018, juli 24). *Supervised vs Unsupervised Learning in 3 Minutes*. Hentet fra Towards Data Science: https://towardsdatascience.com/supervised-vs-unsupervised-learning-in-2-minutes-72dad148f242

Kwiatkowski, R. (2021, mai 22). *Gradient Descent Algorithm — a deep dive*. Hentet fra Toward Data Science: https://towardsdatascience.com/gradient-descent-algorithm-a-deep-dive-cf04e8115f21

Lin, T.-Y., Dollár, P., Girshick, R., Kaiming, H., Hariharan, B., & Belongie, S. (2017). *Feature Pyramid Networks for Object Detection.* CoRR. Hentet fra Hentet fra https://arxiv.org/abs/1612.03144

Lin, T.-Y., Maire, M., Belongie, S., Bourdev, L., Girshick, R., Hays, J., . . . Dollar, P. (2014). *Microsoft COCO: Common Objects in Context.* CoRR. Hentet fra Hentet fra http://arxiv.org/abs/1405.0312

Liu, Y. (2018, Oktober 25). *The Confusing Metrics of AP and mAP for Object Detection / Instance Segmentation*. Hentet fra Webområde for Medium: https://yanfengliux.medium.com/the-confusing-metrics-of-ap-and-map-for-object-detection-3113ba0386ef

Luhaniwal, V. (2019, Mai 7). *Forward propagation in neural networks*. Hentet fra Webområde for Towards Data Science: https://towardsdatascience.com/forward-propagation-in-neural-networks-simplified-math-and-code-version-bbcfef6f9250

MathWorks Inc. (u.å.). *What Is Object Detection?* Hentet fra MathWorks: https://se.mathworks.com/discovery/object-detection.html

Mayo, M. (2018, Oktober 5). *A Concise Explanation of Learning Algorithms with the Mitchell Paradigm*. Hentet fra KDnuggets: https://www.kdnuggets.com/2018/10/mitchell-paradigm-concise-explanation-learning-algorithms.html

Ng, A. (2012). *Machine Learning*. Hentet fra Coursera: https://www.coursera.org/learn/machine-learning/home/week/1

O'Shea, K., & Nash, R. (2015, november 26). *An Introduction to Convolutional Neural Networks.* Hentet fra arXiv.org: https://arxiv.org/abs/1511.08458

Open Robotics. (2018, August 8). *ROS/Introduction*. Hentet fra Webområde for wiki.ros: http://wiki.ros.org/ROS/Introduction

Paul, S. (2018, August). *Hyperparameter Optimization in Machine Learning Models*. Hentet November 15, 2022 fra Datacamp: https://www.datacamp.com/tutorial/parameter-optimization-machine-learning-models

Sahorta, H. (2020, Mars 3). *Google's EfficientDet: An Overview*. Hentet fra Webområde for Towards Data Science: https://towardsdatascience.com/googles-efficientdet-an-overview-8d010fa15860

Sauter, A. (2019). *Skrog [Fotografi].* Sjøkrigsskolen: Upublisert.

Shukla, S. (2022, august 23). *Regression and Classification | Supervised Machine Learning*. Hentet fra geeksforgeeks: https://www.geeksforgeeks.org/regression-classification-supervised-machine-learning/

Stewart, M. (2019, februar 27). *Simple Introduction to Convolutional Neural Networks*. Hentet fra Webområde for Towards Data Science: https://towardsdatascience.com/simple-introduction-to-convolutional-neural-networks-cdf8d3077bac

Tan, M., & Le, Q. V. (2019). *EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks.* CoRR.

Tan, M., Pang, R., & Le, Q. V. (2019). *EfficientDet: Scalable and Efficient Object Detection.* CoRR.

TensorFlow. (2022, September 14). *Object detection*. Hentet fra TensorFlow: https://www.tensorflow.org/lite/examples/object_detection/overview

TensorFlow. (2022, Mai 26). *TensorFlow Lite*. Hentet fra TensorFlow: https://www.tensorflow.org/lite/guide

TensorFlow. (2022, Oktober 27). *TensorFlow Lite Python object detection example with Raspberry Pi*. Hentet fra Github: https://github.com/tensorflow/examples/tree/master/lite/examples/object_detection/raspberry_pi

TensorFlow. (2022, Desember 15). *Transfer learning and fine-tuning*. Hentet fra TensorFlow: https://www.tensorflow.org/tutorials/images/transfer_learning

The Raspberry Pi Foundation. (2016, April). *Raspberry Pi Documentation*. Hentet fra Webområde for The Raspberry Pi Ltd: https://www.raspberrypi.com/documentation/accessories/camera.html

Tsang, S.-H. (2019, Januar 19). *Review: FPN — Feature Pyramid Network (Object Detection).* Hentet fra Webområde for Towards Data Science: https://towardsdatascience.com/review-fpn-feature-pyramid-network-object-detection-262fc7482610

Veronica, A. (2021, januar 15). A Gentle Introduction to Backprpagation and Implementing Neural Network Animation. *Medium.com*.

Vedlegg A – tabell for test av relative peiling

Dette er tabellen over alle målingene som ble gjort under test av relative peiling.

	Grader				
Måling nr.	**5**	**10**	**15**	**20**	**30**
1	5,90	11,09	16,93	22,71	31,15
2	5,96	11,39	16,93	22,24	31,15
3	5,96	11,39	16,99	22,71	31,15
4	5,96	11,39	16,93	22,71	30,80
5	6,31	11,39	16,93	22,24	30,80
6	5,96	11,39	16,93	22,71	30,80
7	5,90	11,39	16,93	22,24	31,15
8	5,84	11,09	16,64	22,24	31,15
9	5,96	11,39	16,93	22,71	31,15
10	6,31	11,39	16,93	22,71	31,15
11	5,84	11,03	16,58	22,24	31,15
12	5,96	11,03	16,93	22,71	31,15
13	6,31	11,03	16,58	22,24	31,15
14	5,96	11,03	16,99	22,71	31,15
15	5,84	11,03	16,93	22,71	31,15
16	5,96	11,03	16,58	22,24	31,15
17	5,96	11,03	16,93	22,71	31,15
18	6,31	11,03	16,93	22,24	31,15
19	5,84	11,39	16,99	22,71	31,15
20	5,96	11,39	16,93	22,71	31,15

Vedlegg B – Operativsystem på RRi_ML og RRi_SV

RPi_SV er den RPi-en som kjører ROS og derav også dronesvermen. For at denne skal fungere blir Ubuntu MATE 18.04.3 benyttet:

https://ubuntu-mate.org/raspberry-pi/

På RPi_ML blir programvaren kjørt på Rasbian 10 Buster:

Operating system images – Raspberry Pi

Vedlegg C – Kildekode lagret i Github

Github er en nettside der man kan lagre kildekode. Det er enkelt å laste ned den nyeste versjonen av koden og samtidig enkelt å laste ned fra en ekstern enhet.

For å kunne laste koden ned trenger man å laste ned git på RPi. For å gjøre dette må følgende skrives inn i Terminal:

sudo apt install git

Dette er et nettområde som er laget for samarbeid på kode for utviklere.

Vi har lagt inn to ulike lenker for denne oppgaven. Den første delen er oppdatert kode i ROS, som kjøres på RPi_SV. Den andre delen er kildekode for deteksjonskonseptet vi har utviklet som kjøres på RPi_ML:

https://github.com/mBachRos/swarm

https://github.com/mBachRos/object_detection_for_drones

I lenke nummer to er det også lagt inn brukergrensesnitt fra Node-Red samt HTML

Dersom det er noen spørsmål til programvaren – Ta kontakt.

Vedlegg D – Guide for å sette opp RPi_ML

For å sette opp RPi_ML er det ulike steg som må gås gjennom. Disse stegene gjennomføres i terminalvinduet på Rasbian OS.

Første steg gjøres ved å oppdatere pakkene på Raspberry Pi operativsystemet:

sudo apt-get update

Sjekk så etter egen Python versjon:

python3 –version

Installer så virtuelt miljø på maskinen din:

python3 -m pip install --user --upgrade pip

python3 -m pip install –user virtualenv

Videre oppretter du et virtuelt miljø for tflite eksemplene:

python3 -m venv ~/tflite

Etter dette aktiverer du det virtuelle miljøet:

source ~~/tflite/bin/activate

Etter dette kloner du fra github ved å skrive følgende:

git clone https://github.com/tensorflow/examples.git

cd examples/lite/examples/object_detection/raspberry_pi

git clone https://github.com/mBachRos/object_detection_for_drones

Etter dette installerer du de nødvendige pakkene:

sh setup.sh

For å starte programmet skriver du følgende:

python detect.py

Dersom feilmeldingen "ImportError: libcblas.so.3: cannot open shared object file: No such file or directory" dukker opp, gjør følgende:

sudo apt-get install libatlas-base-dev

Vedlegg E – QGroundControl

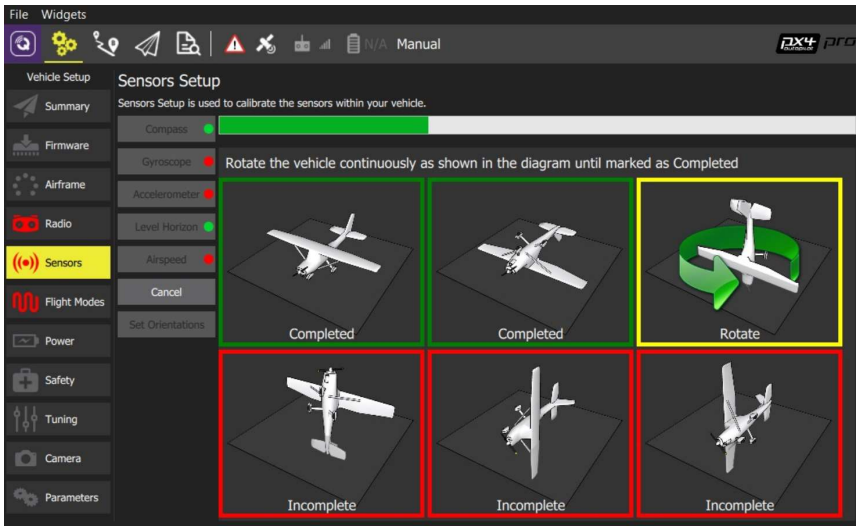
Dette er videreført fra forrige bachelor. QGroundControl blir brukt til å installere firmware til Pixhawkene samt å kalibrere dem når de er nye. Dette har vært brukt for å få sjødronene til å finne rett posisjon samt rett himmelretning. Bildet under viser et bilde fra kalibreringen.

Nyttig informasjon for videreutvikling av sjødronene er at dersom Pixhawkene skal kalibreres, så må dette gjøres med GPS-modulen påsatt skrogdelen som skal benyttes. Videre skal Pixhawk sitte fast i boksen. Når de kalibreres, bør orienteringen settes til YAW-270. Dersom Pixhawken er festet på en annen måte, er det mulig å sjekke orienteringen.

Bilder under er hentet fra:

https://docs.qgroundcontrol.com/master/en/SetupView/sensors_px4.html

Det viser hvordan kalibreringen av kompasset vil se ut.



Nedlastning:

https://docs.qgroundcontrol.com/en/getting_started/download_and_install.html

Oppstart:

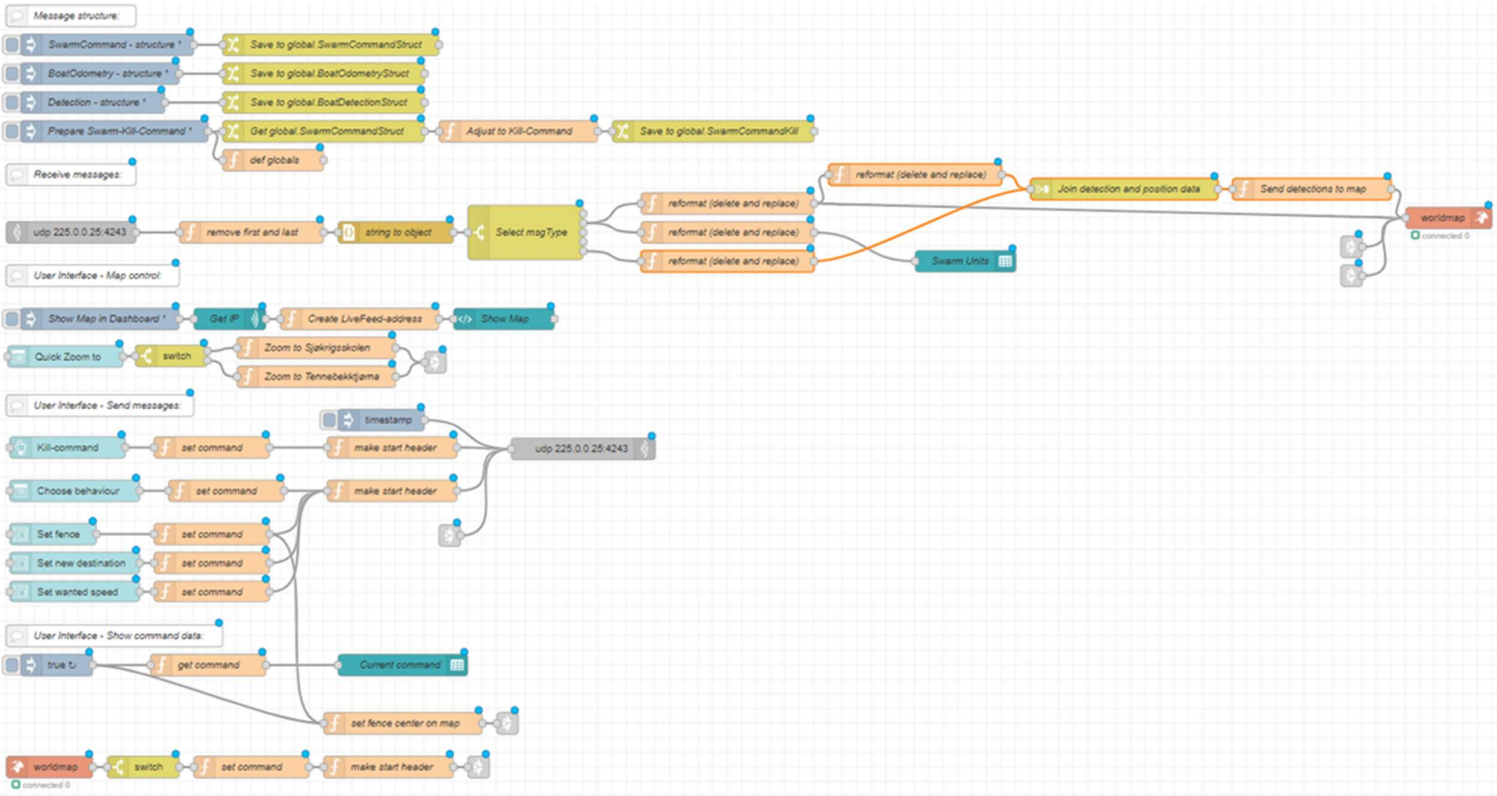
https://docs.qgroundcontrol.com/en/getting_started/quick_start.html

Link til brukerhåndbok:

https://docs.qgroundcontrol.com/en/

Vedlegg F – HMI

Vedlagt er utseende til HMIen i Node-Red. Koden er ved lagt på github som nevnt i vedlegg C.



Vedlegg G – Første test på vannet

Vedlagt er flere tester fra første test på vannet.

Vedlegg H – Andre test på vannet

Vedlagt er flere av bildene fra andre testen på vannet.






Vedlegg I – Resultater fra test av FPS og forsinkelse

Vedlagt er testresultatene fra test av FPS og forsinkelse for RPi3B+ og RPi4.

RPi3	FPS			Forsinkelse	
Målingsnum-mer:	EfficientDet-Lite0	EfficientDet-Lite1	Målingsnum-mer:	EfficientDet-Lite0	EfficientDet-Lite1
1	2.62	1.66	1	2.85	4.78
2	3.61	1.84	2	2.78	4.89
3	3.54	1.78	3	2.61	5.05
4	2.44	1.65	4	2.65	4.67
5	2.92	1.41	5	2.77	3.95
6	3.03	1.38	6	2.79	4.65
7	2.17	1.43	7	2.68	4.71
8	2.33	1.76	8	2.9	4.38
9	2.49	1.76	9	2.59	4.36
10	2.65	1.74	10	2.7	4.44
RPi4	FPS			Forsinkelse	
Målingsnum-mer:	EfficientDet-Lite0	EfficientDet-Lite1	Målingsnum-mer:	EfficientDet-Lite0	EfficientDet-Lite1
1	4.23	2.32	1	1.67	2.77
2	6.22	2.93	2	2.06	2.96
3	6.23	2.95	3	1.54	3.15
4	6.22	2.93	4	1.54	3.16
5	6.25	2.67	5	1.95	2.86
6	6.24	2.75	6	2.08	2.85
7	5.65	2.21	7	2.5	3
8	4.88	2.49	8	1.9	2.94
9	4.35	2.24	9	1.95	2.72
10	5.95	2.49	10	1.48	2.79
Gjennomsnitt	5.622	2.598		1.867	2.92

Vedlegg J – Oversikt over nettverket

Klienter på nettverket			
Passord på alle RPier er: bachelor			
BACHELOR 2019(adresser brukes ikke lengre)			
Navn	**IP**	**RPI**	**OS**
B01	192.168.136.61	B01	MATE
B02	192.168.136.62	B02	MATE
B03	192.168.136.63	B03	MATE
B04	192.168.136.64	B04	MATE
BACHELOR 2022			
HOSTNAME	**IP**	**USER**	**OS**
ROUTER	192.168.136.60	admin	HUAWEI
B05	192.168.136.71	B05	18.04 MATE
B06	192.168.136.72	B05	18.04 MATE
B07	192.168.136.73	B05	18.04 MATE
Ground Control	192.168.136.77	Admin	Windows 10
raspberrypi5	192.168.136.74	pi	Raspbian 10
raspberrypi6	192.168.136.75	pi	Raspbian 10
raspberrypi7	192.168.136.76	pi	Raspbian 10
RPI-PAR			
Swarm Unit	**RPi_ML**	**RPi_SV**	
1	raspberrypi5	B05	
2	raspberrypi6	B06	
3	raspberrypi7	B07	

Vedlegg K – labelImg

Programmet labelImg ble benyttet til merking av alle bilder som ble brukt til trening og testing av maskinlæringsmodellene. Dette ble gjort på en av skolens lab-pcer. Det ble lastet ned med pip i Anaconda og kjørt med samme program.

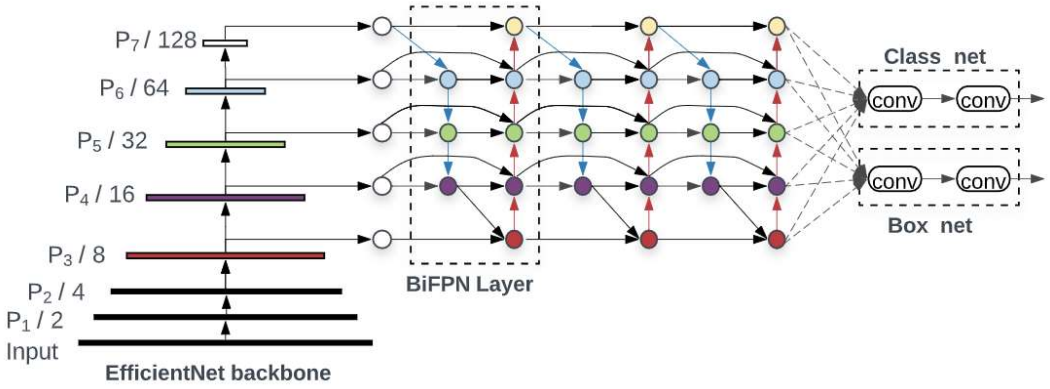
Et utvalg av de merkede bildene fra datasettet er lagt inn i GitHub nettsiden til prosjektet:

https://github.com/mBachRos/object_detection_for_drones

Vedlegg L – Teori om modellene vi bygger videre på

EfficientDet

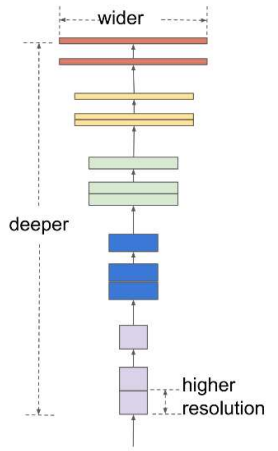
EfficientDet er en gruppe objektdeteksjonsmodeller utviklet av Google Brain Team. (Tan, Pang, & Le, EfficientDet: Scalable and Efficient Object Detection, 2019) Å kjøre objekt-deteksjonsmodeller har tradisjonelt sett vært en vanskelig oppgave for mo-bil- og edge-enheter da disse har begrenset med prosesseringskraft. Men i tillegg til at prosesserings-kraften har økt på disse mindre enhetene så er det gjort store fremskritt i maskinlærings-verden i løpet av få år. I 2019 kom det en gruppe maskinlæringsmodeller til verden ved navn EfficientNet, denne gruppen modeller søkte å minske behovet for prosessorkraft og beholde ytelsen til de beste åpenkilde-modellene. Fra dette arbeidet ble det videreutviklet en gruppe modeller som er laget for objektdeteksjon, disse modellene kalles EfficientDet. Arkitekturen til EfficientDet består av EfficientNet som backbone-struktur, *weighted bi-directional feature pyramid network* (BiFPN) som hals-struktur og to hodestrukturer BoxNet og ClassNet (Tan, Pang, & Le, EfficientDet: Scalable and Efficient Object Detection, 2019). Figur 0-1 viser en oversikt over strukturen til EfficientDet. Videre for-klares hver enkelt del av EfficientDet.



Figur 0-1: Oversikt over EfficientDet strukturen (Tan, Pang, & Le, EfficientDet: Scalable and Efficient Object Detection, 2019)

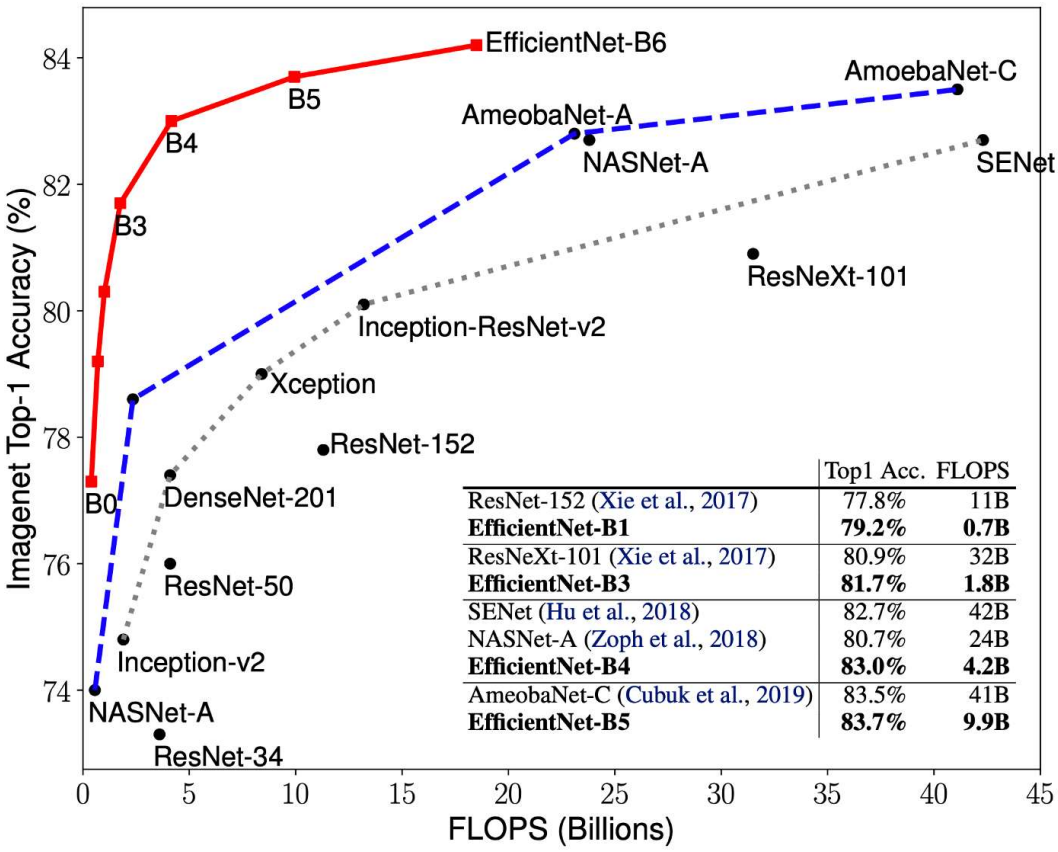
EfficientNet

Backbone-strukturen EfficientNet er et convolutional neural network som ifølge utviklerne av modellene skal være en effektiv og fleksibel strukturå bygge modellen på. Det som gjør EfficientNet effektiv ligger i måten det ble utviklet på, nærmere bestemt en ny metode for skalering av nevrale nettverk. Den vanlige måten å øke et nettverks ytelse på har vært å skalere nettverket ved å enten gjøre nettverket dypere, altså øke antall lag i nettverket. Ved å gjøre det bredere, det vil si å øke størrelsen på filtrene til nettverket. Eller så kan man øke input-bildeoppløsningen, altså hvor mange piksler det kan være i bildene som mates inn i nettverket. Det er også mulig å skalere flere av disse faktorene samtidig for å øke ytelsen, men det har ofte ført til problemer med nettverkets effektivitet. (Tan & Le, EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks, 2019) Figur 0-2 viser løsningen på dette skaleringsproblemet, som var å bruke en konstant rate for å skalere hver av de enkelte dimensjonene dybde, bredde og input-bildeoppløsning. (Tan & Le, EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks, 2019)



Figur 0-2: Tre forskjellige måter å skalere et CNN (Tan & Le, EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks, 2019)

Den nye skaleringsmetoden viste seg å gi gode resultater, men utviklerne ønsket også et effektivt grunnettverk å skalere videre på. For å finne det grunnettverket med høyest presisjon og effektivitet brukte de maskinlæringsalgoritmen *neural architecture search.* Denne algoritmen finner frem til den optimale strukturen for nettverket i forhold til hvilken oppgave nettverket skal løse. Resultatet ble EfficientNet-B0, et nettverk som har relativt få parametere og er lite krevende å kjøre, men som beholdt en nokså god presisjon. Dette nettverket er skalert til syv forskjellige varianter, fra B0 til B6. Figur 0-3 viser hvordan disse har økende ytelse, men også økende behov for prosesseringskraft. (Tan & Le, EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks, 2019)



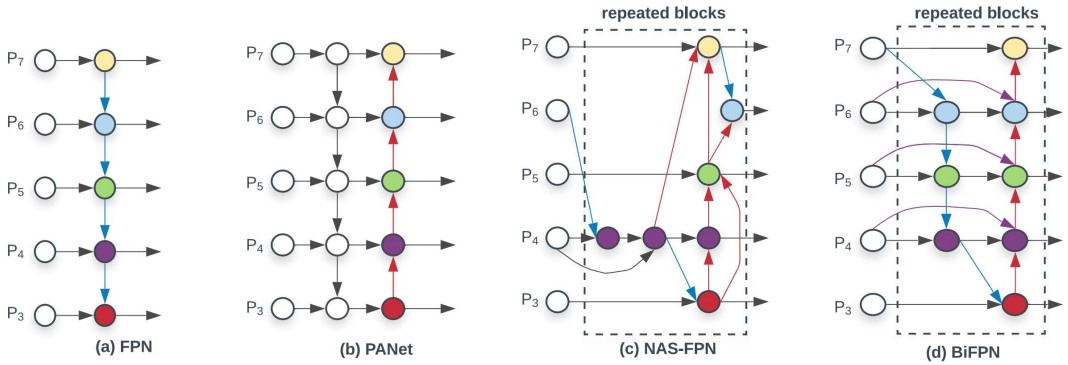
Figur 0-3: Test av EfficientNet modellene på Imagenet datastettet (Tan & Le, EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks, 2019)

BiFPN

Weighted bi-directional feature pyramid network er nettverkshalsen i EfficientDet, den ligger etter EfficientNet-delen som vist på figur 0-1. Med et CNN-nettverk som backbone så kan man benytte et FPN-nettverk som nettverkshals for å fusjonere utgangsnevronene fra CNN lagene. De siste lagene fra backbone-nettverket går videre inn i FPN nettverket som vist på figur 0-1. (Sahorta, 2020)

FPN-nettverket skal kombinere forskjellige utgaver av bildet i forskjellig oppløsning. Et utgangspunkt for å gjøre dette er FPN-nettverksdesign, (a) i figur 0-4, denne fusjonerer nevroner ovenfra og ned. En bedre måte å fusjonere nevroner på kan være PANet, (b) i figur 0-4, som kopler nevroner sammen på en måte som gjør at data kan flyte både ovenfra og ned, og nedenfra og opp. Dette betyr at oppløsningen kan varieres opp og ned når data flyter igjennom nettverket. Et alternativ til PANet er NAS-FPN, (c) i figur 0-4, som ble

utviklet ved bruk av en NAS-algoritme som foreslår et nettverkdesign for best mulig prediksjonsevne med en begrenset mengde nevroner. En kombinasjon av disse to nettverkene ble brukt under uviklingen av BiFPN, (d) i figur 0-4 Forfatterne bak EfficientDet-rapporten fant ut at PANet oppnår en høyere nøyaktighet i prediksjoner enn NAS-FPN-nettverket, men på bekostning av flere parametere og dermed en mer krevende beregningsprosess. (Tan, Pang, & Le, EfficientDet: Scalable and Efficient Object Detection, 2019)



Figur 0-4: Utviklingen av BiFPN (Tan, Pang, & Le, EfficientDet: Scalable and Efficient Object Detection, 2019)

Utviklerne av modellen brukte denne lærdommen og kom frem til en kombinasjon av PANet og NAS-FPN som ble kalt BiFPN. De forenklet PANet ved å fjerne nevroner med bare en input, fordi disse bidro mindre til nevronfusjon. En ekstra kopling ble lagt til fra input-nevroner til output-nevroner på samme nivå, dette bidrar til mer nevronfusjon med en liten kostnad i beregningsprosess. (Sahorta, 2020) Dette resulterte i BiFPN som utgjør halsnettverket til EfficientDet. BiFPN blir skalert med samme skaleringsmetode som utviklerne av EfficientNet benyttet, men med en egen ligning for skalering av bredden og en for skalering av dybden. Ligningene er vist nedenfor.

$$W_{bifpn} = 64 \cdot \left(1.35^{\phi}\right), \qquad D_{bifpn} = 2 + \phi$$

Bredden blir skalert med 64 ganger 1.35 opphøyd i skaleringskonstanten, og Dybden med 2 pluss skaleringskonstanten theta.

EfficientDet-Lite

EfficentDet-Lite er en lettvektsutgave av EfficientDet familien. Det er gjort noen endringer for at det skal kreve enda mindre å kjøre modellene. Disse endringene innebærer at modellene har lavere oppløsning på bilder som sendes gjennom det nevrale nettet. Dette minsker det beregningsmessige behovet for modellen. Eksempelvis er inputoppløsingen til EfficientDet1 640x640, mens EfficientDet-Lite1 bruker 384x384. I tillegg bruker Lite-utgaven en annen aktiveringsfunksjon enn baseutgaven, Sigmoid-funksjonen er byttet ut med en mye enklere aktiveringsfunksjon kalt ReLU6, som vist i nedenfor. (Tan, Pang, & Le, EfficientDet: Scalable and Efficient Object Detection, 2019) ReLU6-funksjonen brukes de matematiske funksjonenes minimumsverdi og maksimalverdi. Dette sørger for at outputverdien blir mellom 0 og 6.

$$ReLU6(x) = \min\left(\max(x, 0), 6\right)$$