Sjøkrigsskolen

Bacheloroppgave

Kombinasjon av objektdeteksjon og AIS for automatisk datainnsamling i utsatte områder

av

Johan A. Bergsund & Magnus W. Polsrød

Levert som en del av kravet til graden:

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

Antall ord: 16 702

Innlevert: 12 2023

Godkjent for offentlig publisering

Publiseringsavtale

En avtale om elektronisk publisering av bachelor/prosjektoppgave

Kadetten(ene) 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 kadetten(ene) har godkjent publisering.

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

Jeg (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

Jeg (Vi) erklærer herved at oppgaven er mitt eget arbeid og med bruk av riktig kildehenvisning. Jeg (Vi) har ikke nyttet annen hjelp enn det som er beskrevet i oppgaven.

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

Dato: 02-11-2023

Johan Anderssen Bergsund	Johan A. Bergsund
Kadett navn	Signatur
Magnus Wedege Polsrød	Magnus W. Polsrød
Kadett navn	Signatur

Forord

Denne bacheloroppgaven er skrevet gjennom høstsemesteret 2023 ved sjøkrigsskolen, og er innlevert som sluttoppgave for graden «Bachelor i militære studier, med fordypning i ledelse og marineingeniør våpen, elektronikk og data». Oppgaven er skrevet av Johan Anderssen Bergsund og Magnus Wedege Polsrød, begge tilhørende kull 2020-2023 ved FHS Sjøkrigsskolen.

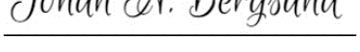
Prosjektet tar for seg datainnsamling, og utforsker hvordan en kombinasjon av objektdeteksjon med kamera og automatisk identifikasjonssystem (AIS) kan brukes til datainnhenting i maritime områder. Implementeringen og testetingen av systemet har i sin helhet blitt utført på og ved Sjøkrigsskolen i Bergen.

Arbeidet med prosjektet har vært givende og meget lærerikt, da begge hadde lite erfaring med temaene som er utforsket i oppgaven. Det har blitt brukt mye tid på å sette seg inn i teori og systemer, både nye systemer for dette prosjektet og systemer brukt i tidligere bacheloroppgaver. Oppgaven reflekterer så mange aspekter ved prosjektet som mulig, uten å bli for dyptgående på detaljplan. Det forutsees derfor at leseren har grunnleggende teknisk kompetanse, samt kunnskap om kunstig intelligens (KI) og maskinlæring.

Vi ønsker å gi en stor takk til førsteamanuensis Christophe Massacand og førsteamanuensis Alexander Sauter, for konkret og viktig veiledning gjennom hele perioden. Videre ønsker vi også å takke tidligere kadetter ved Sjøkrigsskolen Eivind Horten, Sindre Ytterstad, Wodel Rugahiye, samt Olav Hollup og Magnus Svensson Røsand, for behjelpelighet med problemløsning og spørsmål under arbeidsprosessen.

Bergen, Sjøkrigsskolen, 02-11-2023

Johan Anderssen Bergsund



Magnus Wedege Polsrød

Sammendrag

Norge, med den langstrakte og eksponerte kysten, står overfor potensielle utfordringer med å opprettholde tilstrekkelig situasjonsforståelse over sine maritime områder. Fysisk overvåkning og tilstedeværelse på sjøen krever store ressurser innenfor materiell, mannskap og tid. Dette er noe som belyser den potensielle resursgevinsten automatiserte løsninger kan medføre i maritim overvåkning. Oppdaterte overvåkningsmetoder spiller en sentral rolle for sikker skipsfart, samt for nasjonal sikkerhet gjennom understøttelse av militære operasjoner. Ved å innhente og sammenfatte data fra forskjellige kilder langs kysten, kan man potensielt skape innholdsrike datasett for spesifikke maritime lokasjoner. Denne oppgaven har derfor undersøkt problemstillingen:

Hvordan kan man designe og implementere et system som detekterer og samler inn data fra fartøy i utsatte maritime områder, slik at denne dataen kan brukes til spesialiserte datasett for fremtidige maskinlæringsapplikasjoner?

Systemet i denne oppgaven benytter seg av informasjon i form av bilder tatt gjennom en algoritme for objektdeteksjon, og AIS-meldinger. Systemet er også utviklet til å være et mobilt system som potensielt skal kunne utplasseres på forskjellig lokasjoner, dermed er konseptet i stand til å etablere miljøspesialiserte datasett. Systemet kan, med sin evne til også å identifisere fartøy som mangler AIS-data, bidra til å oppdage skip som potensielt kan være en sikkerhetstrussel.

Gjennom utviklingen av konseptet har systemets funksjonalitet blitt omfattende testet. Disse testene viser at systemet er i stand til å detektere fartøy, og lagre deteksjonene i form av bilder, tilhørende bildeinformasjon og med tilgjengelig AIS-data. For å bekrefte systemets potensiale opp mot problemstilling, har det blitt trent og testet en maskinlæringsmodell basert på innhentet data fra systemet. Resultatene fra testene reflekterer at systemet er i stand til å hente inn gode bilder til bruk for maskinlæringsmodeller. Videre konkluderes det med at systemet, med riktig bruk, kan være et verdifullt tilskudd til maritim overvåkning nå og i fremtiden.

Innholdsfortegnelse

Figurer og formler

Tabeller og diagrammer

Nomenklatur

AI - Artificial Intelligence

AIS - Automatic Identification System

API - Application Programming Interfaces

BLOB - Binary Large Object

DB4S - DB Browser for SQLite

FOV - Field of View

FPN - Feature Pyramid Network

GNSS - Global Navigation Satellite Systems

HQ - High Quality

ReLU - Rectified linear activation function

RPI - Raspberry Pi

SQL - Structured Query Language

Maskinlæring - spesialisering innen kunstig intelligens hvor det brukes metoder for å la datamaskiner finne mønstre i store datamengder (Tidemann & Elster, 2023)

Datasett - et datasett er en samling med data. Dataen kan bestå av forskjellig type data og for eksempel lagres/fremstilles i form av en database

1 Introduksjon

I introduksjonskapittelet presenteres oppgavens rammeverk som innebefatter bakgrunn, problemstilling, mål, avgrensning og struktur.

1.1 Bakgrunn

Maritim overvåkning, eller overvåkning generelt, går ut på å «[...] observere, registrere og/eller lagre informasjon om objekter, fenomener, grupper og/eller individer» (Løkke, 2023). Det finnes mye man kan overvåke i og ved sjøen, som dyreliv og værforhold, men i denne oppgaven brukes begrepet «maritim overvåkning» om systematisk informasjonsinnhenting om skipstrafikken på sjøen. Maritim overvåkning er viktig både for den sivile og militære sektoren. Sivile grunner for å drive overvåkning kan være sikker navigasjon, miljøbeskyttelse, sikre fremkomst i de viktigste handelsrutene og håndhevelse av lover (Kystverket, 2019). I militær sektor er det avgjørende for nasjonal sikkerhet at vi driver overvåking av potensielle trusler i våre egne territorialfarvann (FFI, u.d.). Tradisjonelt har maritim overvåking vært avhengig av manuelle prosesser i form av fysisk tilstedeværelse, noe som ikke nødvendigvis klarer å holde tritt med kompleksiteten og utviklingen i trafikkbildet på havet. Et eksempel på dette er å faktisk ha fartøy med besetning ute på sjøen. Dette er noe som naturlig nok medfører begrensninger innenfor ressurser og tilstedeværelse på rett sted til rett tid. Et annet eksempel på overvåking er automatisk identifikasjonssystem (AIS), dette er et system for å sende og motta informasjon om fartøy og er påbudt å bruke for sivile fartøy (Kystverket, 2019). Dette er derimot et system som forutsetter at fartøy faktisk sender ut disse meldingene korrekt og ikke ønsker å skjule noe, som igjen gjør at man *alene* ikke kan stole på denne datakilden for optimal overvåking.

Veksten i global skipstrafikk forventes å fortsette i årene fremover, noe som kan gjøre det mer utfordrende å overvåke sjøtrafikken (Kystverket, 2020). Med en forventet trafikkøkning på rundt 40 prosent i norske farvann innen 2040, er det en tilsvarende økning for at potensielle maritime ulykker og trusler kan gå udetektert (Kystverket, 2020). Som respons har Kystverket satt i verk tiltak for å forbedre sjøsikkerheten, inkludert utvidelse av sjøtrafikksentralenes dekningsområde og utbygging av AIS-nettverket. Disse tiltakene er kritiske for å møte fremtidige utfordringer knyttet til trafikkvekst, digitalisering og klimaendringer, noe som understreker behovet for teknologisk innovasjon for å opprettholde og forbedre sjøsikkerhetsnivået langs kysten (Kystverket, 2020).

«Oppdatert overvåking står sentralt i militære operasjoner, på hele skalaen fra fred og kriser til full krig. Det overvåkes på land, i luft, sjø, verdensrommet og cyberspace» (FFI, u.d.). Fra et sjømilitært ståsted «[...] kan det være plausibelt å anta at dagens styrkestruktur ikke gir tilstrekkelig situasjonsforståelse langs kysten til enhver tid.» (Hollup & Røsand, 2022). Som beskrevet i utdragene over, er overvåkning viktig og vanskelig. Inntoget av maskinlæringsmodeller representerer en forandring i denne sektoren, da overvåkning kan gjøres automatisert uten fysisk tilstedeværelse. Disse modellene har potensialet til å identifisere fartøy, noe som er avgjørende for å opprettholde situasjonsforståelse på sjøen.

For å kunne bruke automatiserte løsninger som maskinlæringsmodeller, er man avhengig av data som representerer hva disse modellen skal klare å kjenne igjen. Informasjonsinnhenting kan skje ved bruk av mange metoder, og forskjellig data kan samles inn. Eksempelvis kan maskinlæringsmodeller spesialtilpasses til å identifisere med grunnlag i bildeinformasjon. «FFI er med på å utvikle systemer som gjør det lettere å sammenstille og analysere data fra mange overvåkingsplattformer svært raskt» (FFI, u.d.). Forsvarets forskningsinstitutt (FFI) jobber allerede med systemer for sammenfatning av data fra forskjellige kilder, men før dette steget behøves ideelt sett store datasett. For eksempel kan mange skip oppfattes med lik form og størrelse, videre er også forholdene i og rundt kysten varierende fra sted til sted. Dette understøtter det sentrale behovet for store datasett, da man må ha mye data om hvert enkelt skip for å potensielt kunne trene en modell til å identifisere unike skip på forskjellige steder.

I Hollup og Røsand sin bacheloroppgave fra 2022, utviklet de et konsept for maritim overvåkning ved bruk av maskinlæringsmodeller til objektdeteksjon på droner (Hollup & Røsand, 2022). Målet med denne oppgaven var å detektere et spesifikt fartøy, og de skrev følgende i sin konklusjon; «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.» (Hollup & Røsand, 2022). Dette understreker viktigheten av at datasett for maskinlæring ikke bare burde være store, men også må hentes inn fra de ønskede operasjonsområdene.

FFI, Kongsberg Norcontrol og Kystverket samarbeider allerede på et forsknings- og utviklingsprosjekt ved navn BEAN. BEAN skal bli et system for automatisert overvåkning for å øke sikkerheten langs den norske kyst (Kystverket, 2020). Dette er et prosjekt under utvikling, og er ikke ferdigstilt. Et slikt overvåkningssystemet satt i kontekst med at FFI

allerede jobber med et system for å sammenfatte og analysere data fra forskjellige kilder, belyser behovet for innhenting av store datasett i forkant. Med bakgrunn i konklusjonen til Hollup og Røsand er det også viktig og helt sentralt at datasettene inneholder data fra forskjellige miljøer for optimalisering av disse systemene.

1.2 Problemstilling

Med bakgrunnen som utgangspunkt er problemstillingen til denne oppgaven som følger:

Hvordan kan man designe og implementere et system som detekterer og samler inn data fra fartøy i utsatte maritime områder, slik at denne dataen kan brukes til spesialiserte datasett for fremtidige maskinlæringsapplikasjoner?

Med dette som problemstilling eller forskningsspørsmål, undersøker denne oppgaven design og implementering av et slikt system. Systemet skal være spesielt tilrettelagt for å hente inn store og unike datasett i områder av interesse.

1.3 Mål

Systemet som skal utforskes i denne oppgaven er et system som samler inn data fra to kilder, disse er deteksjonsdata fra et kamera og AIS-meldinger. Systemet består av flere komponenter, og det er derfor hensiktsmessig å konkretisere noen mål for prosjektet. Målene belyser systemets funksjonalitet, og forklarer de generelle kravene vi stiller til systemet. Intensjonen med prosjektet er å lage et mobilt system, som kan bidra til maritim overvåking og datainnhenting i områdene man ønsker å utplassere systemet. Målene er lagt frem i tabell 1-1.

Mål	Beskrivelse
Datainnsamling	Kunne automatisk detektere og lagre deteksjons-data, samt kontinuerlig innhente AIS-data i et avgrenset område.
Sammenligning/identifikasjon	Dataen fra de forskjellige kildene skal kunne håndteres sammen for identifikasjon, samt iden-tifisere deteksjoner som mangler AIS-data.
Database	Datapakker med relevant informasjon om en de-teksjon skal lagres i en database.
Visualisering/overvåkning	Det skal være mulig for en eventuell operatør å se når nye datapakker kommer inn i databasen, samt overvåke systemets drift.

Tabell 1-1: Mål

1.4 Begrensninger

I prosjektet har vi benyttet oss av systemer på to separate enheter, noe som medførte at vi i dette konseptet brukte internett for å kommunisere mellom enhetene. Dette begrenser mobiliteten i form av rekkevidde, men siden mobilitet ikke er hovedmålet har det ikke blitt prioritert å finne den mest optimale løsningen på akkurat dette punktet. Det har like-vel blitt prøvd å tilrettelegge for et så mobilt produkt som mulig på andre områder. I tillegg hentes AIS-dataen inn fra en database som også avhenger av internett, som bidrar til en begrensning innenfor internettdekning i denne oppgaven.

Som nevnt hentes AIS-dataen fra en database. Databasen er åpen og tilgjengelig for alle, noe som medfører begrensninger i tilgjengelig data. Det finnes også en server med mer detaljert og hyppigere datasending, for å få tilgang til denne krever det en innvilget søk-nad til Kystverket og tilgang på en fast IP-adresse. Vi fikk innvilget søknad, men hadde problemer med tilgang på en fast IP-adresse. Det ble derfor ikke prioritert videre da den offentlige IP-adressen i dette tilfellet vil fungere som 'proof of principle', altså at konsep-tet i seg selv fungerer.

Trening av maskinlæringsmodeller er en ressurs- og tidkrevende prosess, og bruken av disse modellene i vår oppgave er derfor begrenset til åpenkildekode. Det er ikke prioritert å tilpasse helt egne maskinlæringsmodeller, derfor er det stort sett brukt helt ferdigtrente modeller med noe videretrening av en modell for å demonstrere konseptets virkemåte.

Systemet bruker kun et heldigitalt kamera som deteksjonssensor, noe som medfører avhengighet av lys. Dette blir derfor en begrensning innenfor testing av systemet, da det må skje med dagslys. Det må heller ikke være dårlig sikt i from av for eksempel tåke. Implementering av andre sensorer, som termisk kamera, muliggjør bruken av systemet uten lys og godt vær.

Mobilitet har blitt prioritert på noen områder utenom kommunikasjon, da dette er et viktig kriterium for utplassering av produktet i prosjektet. Systemets kapasitet vil derfor bære preg av at vi har brukt mobile enheter. Med kraftigere maskinvare kunne presisjonen og konfidensnivået på objektdeteksjon økt, men da på bekostning av mobilitet. Foreløpig begrenser derfor oppgaven seg til prosessorkraften til front-end enheten, som er en Raspberry Pi.

Avslutningsvis er tid og budsjett også begrensninger for dette prosjektet. Vi har begrenset med tid til å gjennomføre prosjektet, noe som igjen gjør at vi har måttet prioritere hvilke aspekter av prosjektet som er viktigst for gjennomføring. Videre setter også budsjettrammen begrensinger for hvor avansert utstyr som kan kjøpes inn.

1.5 Struktur

Oppgaven struktureres i seks hovedkapitler. I innledningen settes rammene for oppgaven ved å beskrive bakgrunnen for valg av tema, definere forskningsspørsmålet, og forklare oppgavens mål og avgrensninger. Deretter presenteres det nødvendige teoretiske grunnlaget som støtter vår forskning i teoridelen. Oppgaven fokuserer spesielt på aspekter av maritim overvåkning, maskinlæring og tilhørende plattformer.

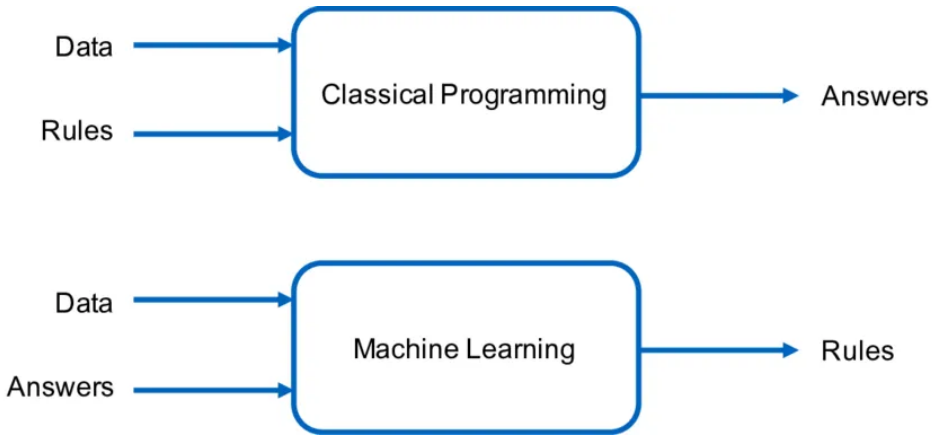
I kapittel tre kommer beskrivelsen på dette prosjektets implementering av den tekniske løsningen. Deretter tar oppgaven for seg resultatanalysen hvor vi presenterer og analyserer resultater om systemets ytelse, inkludert presisjonen av objektdeteksjon og datainnsamling. I kapittel fem, drøfting, blir den tekniske løsningen drøftet opp mot resultater, samt potensiale konseptet kan ha for maritim overvåking. Til slutt vil det siste kapittelet avslutte oppgaven med en konklusjon, og gi anbefalinger til videre arbeid.

2 Teori

I dette kapittelet vil vi belyse nødvendig teori for å danne et forståelsesgrunnlag for hvordan prosjektets system har blitt implementert. Kapittelet starter derfor med å ta for seg teori om maskinlæring og objektdeteksjon. Kapittelet avsluttes med teori om AIS-data, en gjennomgang av plattformer for datainnhenting og lagring, samt kommunikasjon mellom enheter.

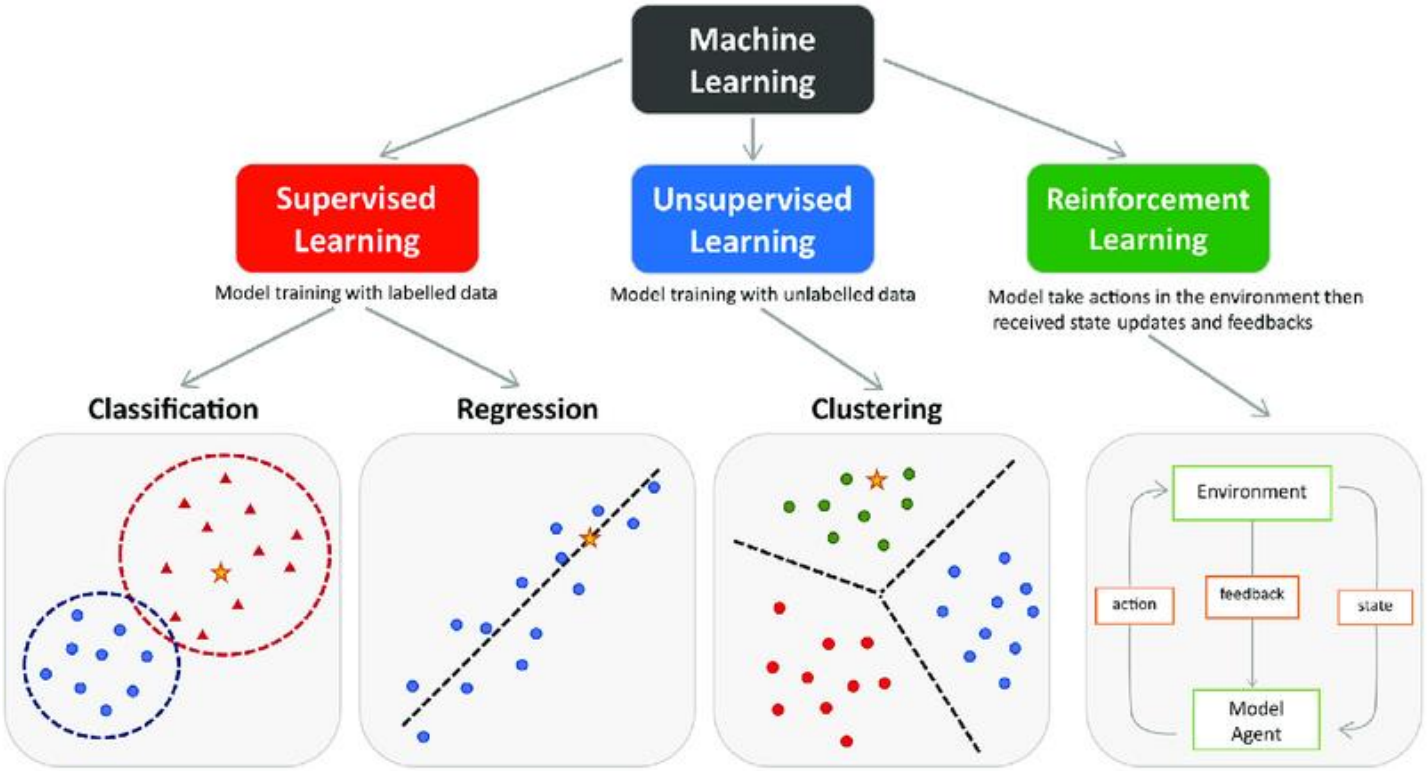
2.1 Maskinlæring

Maskinlæring er en spesialiseringsgren innen kunstig intelligens (KI), hvor maskiner anvender statistiske metoder for å identifisere mønstre i store datasett. I motsetning til tradisjonell programmering, hvor maskiner utfører forhåndsprogrammerte instruksjoner, er målet med maskinlæring at maskinene skal "lære" å gjenkjenne mønstre og ta beslutninger basert på dataene de analyserer uten å spesifikt være programmert til å gjøre det. Dette tillater maskinene å utvikle evnen til å utføre oppgaver og løse problemer «på egenhånd», basert på den erfaringen og kunnskapen de tilegner seg gjennom læreprosessen (Tidemann & Elster, 2023).



Figur 2-1: Figur som beskriver forskjellen på klassisk programmering og maskinlæring. (Papaemmanouil, 2022)

Som beskrevet i teksten og figur 2-1 er maskinlæring avhengig av en læreprosess før den blir tatt i bruk. I maskinlæring gjøres dette hovedsakelig på tre forskjellige måter, veiledet læring (Supervised Learning), ikke-veiledet læring (Unsupervised Learning) og forsterket læring (Reinforcement Learning), (Tidemann & Elster, 2023).

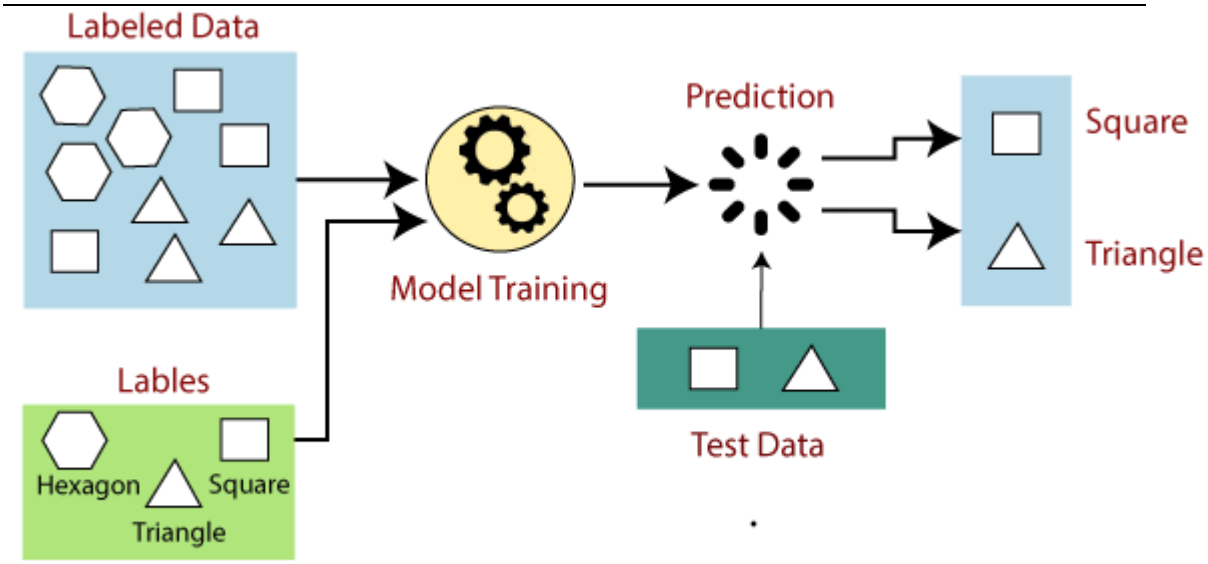


Figur 2-2: En overordnet struktur over de hovedtilnærmingene i maskinlæring. (Peng, Jury, Dönnes, & Ciurtin, 2021)

2.1.1 Veiledet læring (Supervised Learning)

Veiledet læring, eller overvåket læring, er en sentral metode innen maskinlæring hvor modeller utvikles ved å lære fra datasett som allerede er merket med korrekte svar. Dette er maskinlæringsmetoden som er brukt i prosjektet. I denne prosessen, blir inndataene i datasettet presentert for modellen sammen med de tilsvarende etikettene eller "utdataene". Dette gir modellen en klar referanse for hva den skal lære å gjenkjenne eller forutsi (Baheti, 2021).

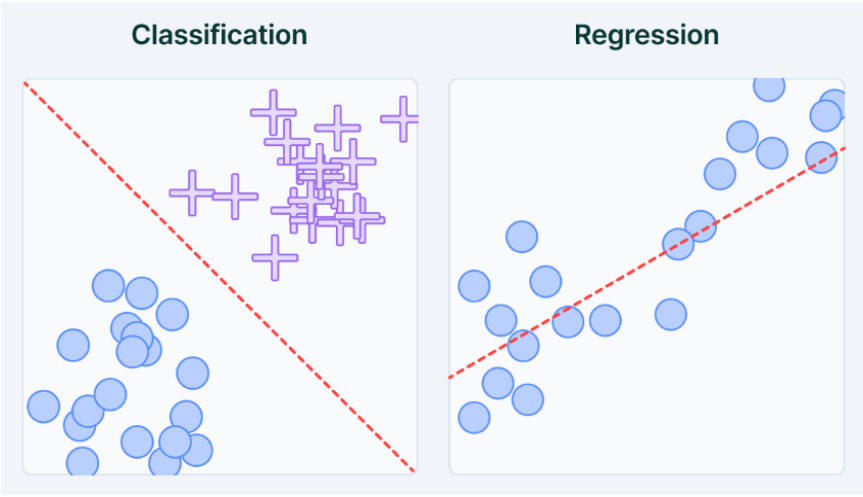
Prosessen starter med at modellen analyserer treningsdataene, som består av både inndata og kjente utdata. Ved å bruke disse merkede dataene, lærer modellen å assosiere bestemte innganger med spesifikke utganger. Denne treningsfasen er kritisk, da den etablerer grunnlaget for modellens evne til å gjøre nøyaktige prediksjoner (Ng, 2012). Videre testes modellen ved å bytte ut inngangsdataen med data den aldri har sett før (testsett), dette sier noe om hvor god modellen er til å kjenne igjen mønsteret den har lært seg. Figur 2-3 under fremstiller veiledet læring.



Figur 2-3: Veiledet læring (Supervised learning). Hentet fra (Supervised Machine Learning, u.d.)

Regresjon i konteksten av maskinlæring refererer til en type algoritme som fokuserer på å forutsi kontinuerlige verdier, i motsetning til diskrete kategorier. Dette kan for eksempel være forutsigelse av boligpriser, aksjekursers eller temperaturmålinger (Ng, 2012).

Klassifisering handler om å ta en inngangsverdi og kartlegge den til en diskret verdi. Dette betyr at modellen tar inndata og avgjør hvilken kategori eller klasse disse dataene tilhører. I klassifiseringsproblemer består utdataene typisk av ulike klasser eller kategorier. Disse klassene er forhåndsdefinerte og utgjør de ulike mulige kategoriene som inndataene kan tilordnes, for eksempel er dette sentralt i bildeklassifisering og objektgjenkjenning hvor modellene søker etter å gjenkjenne disse klassene (Ng, 2012). Figur 2-4 illustrer regresjon og klassifisering.

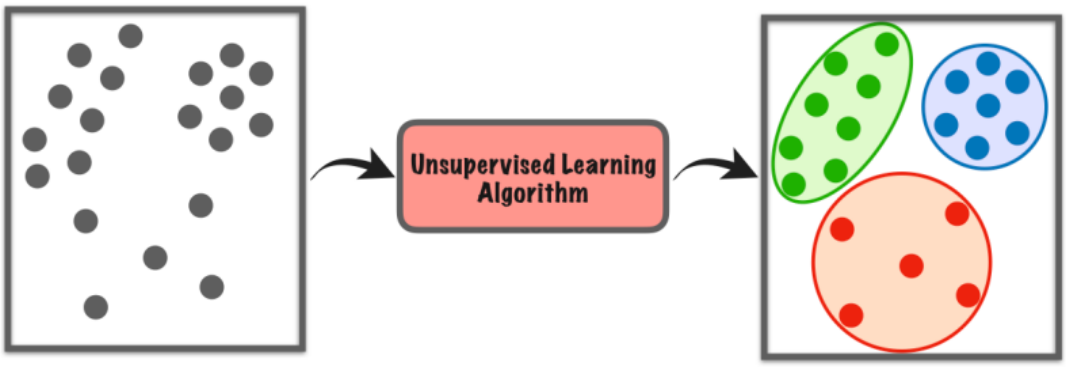


Figur 2-4: Illustrasjon av klassifisering og regresjon. (Baheti, 2021)

2.1.2 Ikke-veiledet læring (Unsupervised Learning)

Ikke-veiledet læring og forsterket læring som figur 2-2 presenterer, er to former for maskinlæring som ikke brukes i oppgaven. Det er fortsatt nyttig for oppgaven å presentere ikke-veiledet læring, da det belyser bruksområdene veiledet læring passer til og hva det ikke er egnet til. Som navnet antyder, skiller ikke-veiledet læring seg fra veiledet læring. I veiledet læring kreves det merkede utdata som korresponderer med inndataene for å trene en modell. Imidlertid er det ikke alltid vi har tilgang til merkede utdata. I slike tilfeller blir det av interesse å anvende teknikker som kan avdekke skjulte mønstre i datasettet. Ikke-veiledet læring er nettopp en slik metode eller teknikk som tilbyr en løsning for å utforske og forstå datasett der markert utdata, eller klare svar, ikke er tilgjengelig (Ng, 2012)

Ved bruk av denne maskinlæringsmetoden får modellen inndata som ikke er markert, videre får modellen muligheten til å behandle dataen på egenhånd. Denne metoden sammenlignes med hvordan hjernen til mennesker lærer nye ting, når modellen får data er målet at den skal klare å finne det underliggende mønsteret og strukturere det etter likheter den finner i dataen (Baheti, 2021).



Figur 2-5: Ikke-veiledet læring. (Jeffares, 2018)

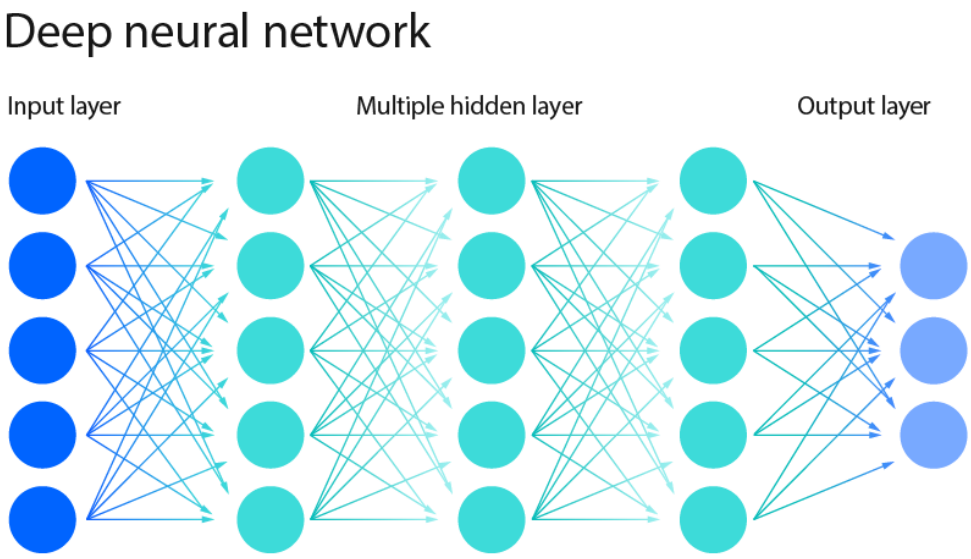
Klyngedannelse (clustering) er en type uovervåket læring der finner skjulte mønstre i data basert på deres likheter eller forskjeller. Som illustrert i figur 2-5 grupperes objektene inn i grupper med de andre objektene de har mest likheter med, og separeres fra de som de har mindre eller ingen tilknytning til. Disse mønstrene kan være relatert til form, størrelse, eller farge, og brukes til å gruppere dataelementer eller skape klynger. Det finnes flere typer klyngedannelsesalgoritmer, som eksklusiv, overlappende, hierarkisk, og sannsynlighetsbasert (Baheti, 2021).

En annen hyppig brukt metode i ikke-veiledet læring er assosiasjon (association), og refererer til å finne relasjoner mellom innholdet i store databaser. Denne metoden bestemmer hvilke sett av elementer som ofte forekommer sammen i datasettet. Denne typen maskinlæring er ofte brukt i bedrifter, da det kan gjøre markedsføringsstrategier mer effektive ved å avdekke slike sammenhenger. For eksempel kan det være at kunder som kjøper ett produkt, også ofte har større sannsynlighet for å kjøpe et annet produkt (Baheti, 2021).

2.1.3 Nevrale nettverk og dyplæring

Nevrale nettverk, også kjent som kunstige nevrale nettverk eller simulerte nevrale nettverk, er en underkategori av maskinlæring og utgjør kjernen i algoritmer for dyplæring. Både navnet og strukturen til nevrale nettverk er inspirert av sammensetningen til det menneskelige eller biologiske nervesystemet/hjernen, da kommunikasjonen skjer på samme måte som de biologiske nevronene snakker med hverandre (Ng, 2012).

Oppbygningen av nevrale nettverk består av lag med noder, som inneholder et inngangslag (input layer), ett eller flere skjulte lag (hidden layer) og et utgangslag (output layer). Hver node i et lag er koblet til alle nodene i det neste laget, dette er det som gjør at det blir et nettverk med mange kommunikasjonslinjer. Hver node i et lag er tilknyttet vekter og én aktivasjonsfunksjon, som gir ut en aktivasjonsverdi. Alle aktivasjonsverdiene fra dette laget er så overført til nodene fra neste lag, som da kombinerer disse verdiene med sine egne vekter og aktivasjonsfunksjoner for å så beregne ny aktivasjonsverdi (IBM, 2023).



Figur 2-6: Fremstilling av nevrale nettverk. (IBM, 2023)

For å oppsummere kan man se på hver enkelt node i et slikt nettverk som en klassifikasjonsmodell som er trent og behandler data. Som sagt inneholder nodene flere input, flere vekter, en aktivasjonsfunksjon og en output. Når et inngangslag er bestemt, tildeles nodene i dette laget vekter. Disse vektene bidrar til å bestemme viktigheten av hver enkelt variabel, med større vekter blir betydeligheten og bidraget til utdataene mer betydelige sammenlignet med andre inndata. Deretter blir utdataene sendt gjennom en aktivasjonsfunksjon. Hvis utdataen aktiverer aktivasjonsfunksjonen, sendes dataen til neste lag i nettverket (IBM, 2023).

Dyplæring er et kjent begrep i maskinlæring og referer til konseptet «dype nevrale nettverk». Begrepene dyplæring og nevrale nettverk blir ofte brukt om hverandre, derfor er det verdt å merke seg at "dyp" i dyplæring bare refererer til antall lag i et nevralt nettverk. Som man ser på figur 2-6 kan et nevralt nettverk inneholde tilnærmet uendelig mange skjulte lag. Et nevralt nettverk som består av mer enn tre lag kan betraktes som en dyplæringsalgoritme, men et nevralt nettverk som bare har to eller tre lag, er bare et grunnleggende nevralt nettverk (IBM, 2023). I teorien kan dype nevrale nettverk kartlegge alle typer inndata til alle typer utdata. Men det er viktig å være bevisst på at denne metoden også krever mye trening sammenlignet med andre maskinlæringsmetoder. Det kreves store datasett, tid og ikke minst maskinvare som kan håndtere det (Ng, 2012).

2.1.4 Datasett og trening

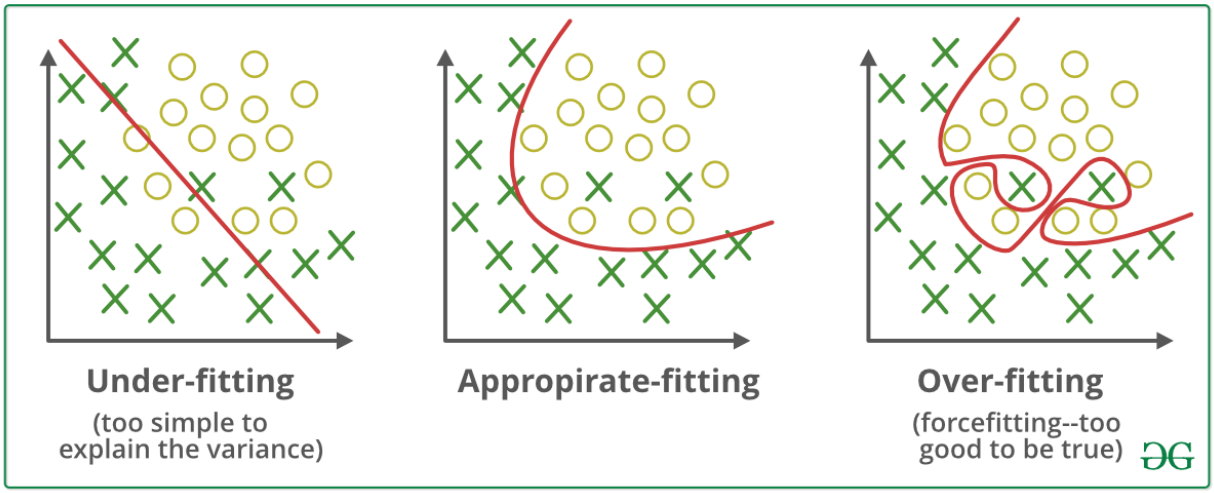
Datasett er viktig for treningen av maskinlæringsmodeller. Datasettene består av en samling av data som er organisert på en oversiktlig måte. Slike datasett kan for eksempel bestå av bilder, men også annen type data som man ønsker å trene modellen på. Hver av disse dataene inneholder ofte attributter eller beskrivelser som korresponderer med informasjonen, og dette legger selve grunnlaget for trening og validering av maskinlæringsmodeller (Tensorflow, 2023).

Trening av maskinlæringsmodeller kan være ressurskrevende, spesielt når vi snakker om trening av dype nevrale nettverk. Grunnen til dette er at det krever store datasett for å oppnå ønsket resultat, og for å unngå over-fitting, som er forklart under. Store datasett kan også være tidkrevende, da denne dataen som regel må prosesseres før treningen, spesielt når det gjelder innhenting og markering av data. Markering er spesielt tidskritisk når dette må gjøres manuelt, prosessering derimot er bare tidskritisk hvis det må gjøres ofte eller hvis nettverket skal kunne brukes veldig raskt etter treningen. Selve innhentingen av data kan variere når det kommer til tid, noen datakilder finnes det mye av, mens ved andre

områder må man først finne/samle data for å ha noe som helt til et treningssett. Når vi bruker begrepet «ressurskrevende», er det ikke bare menneskekraft og tid som står i fokus, men også maskinvare. GPUer (Graphics Processing Units) spiller en kritisk rolle i å akselerere treningsprosessen av maskinlæringsmodeller. GPUer er spesielt effektive for parallelle beregninger, noe som er en grunnleggende del av treningen av dype nevrale nettverk. Det å ha tilgang på nok maskinvare kan være både fysisk vanskelig og dyrt (Tensorflow, 2023).

Som beskrevet over er det å trene en modell helt fra bunn en ressurs- og tidskrevende prosess. Det er derfor vanlig å benytte seg av overføringslæring, videretrening eller som det heter på engelsk «Transfer learning». Dette er et konsept som belager seg på å benytte ferdigtrente modeller, får å så tilpasse disse videre for å fungere spesifikt for sitt eget formål. Dette gjør at datasettene man trenger for å tilpasse modellen kan være mye mindre, og denne dataen blir integrert inn i den ferdigtrente modellen som er lagd i en mye mer omfattende prosess. Et eksempel som er relevant for denne oppgaven er bildekategorisering eller bildegjenkjenning. Hvis en modell trenes på et stort og generelt nok datasett med bilder, vil denne modellen effektivt fungere som en generell modell av den visuelle verden. Deretter kan man dra nytte av disse lærte algoritmene uten å måtte starte fra bunnen av med å trene en stor modell på et stort datasett, men tilpasse den til å bare kjenne igjen spesifikke objekter (Tensorflow, 2023).

Ved trening av modeller er det mange parametere som må overvåkes og justeres for å oppnå et så godt resultat som mulig. To begreper man bruker i trening er undetilpasning (underfitting) og overtilpassning (overfitting), og dette er to utfall som er uønskede når man trener en maskinlæringsmodell. En maskinlæringsalgoritme sies å ha undetilpasning når en modell er for enkel til å fange opp kompleksiteten i datasettene den får presentert. Dette sier noe om modellens manglende evne til å lære treningsdataene effektivt, noe som resulterer i dårlig ytelse. Enkelt sagt resulterer dette i modeller som er unøyaktige, spesielt når de anvendes på nye og usette eksempler. En modell kan derimot også være overtilpasset når den ikke gjør nøyaktige prediksjoner når den brukes på testdata. Når en modell trenes med for lite data eller at modellen man har valgt er unødvendig kompleks, er det en risiko for at modellen blir overtilpasset (over-fitting). Det betyr med andre ord at modellen lærer seg veldig spesifikke detaljer fra treningsdataen, noe som ikke nødvendigvis er realistisk for det modellen skal anvendes til (Dewang, 2023). Figur 2-7 illustrer under-, riktig- og overtilpassning.



Figur 2-7: Undetilpasning og overtilpassning. (Dewang, 2023)

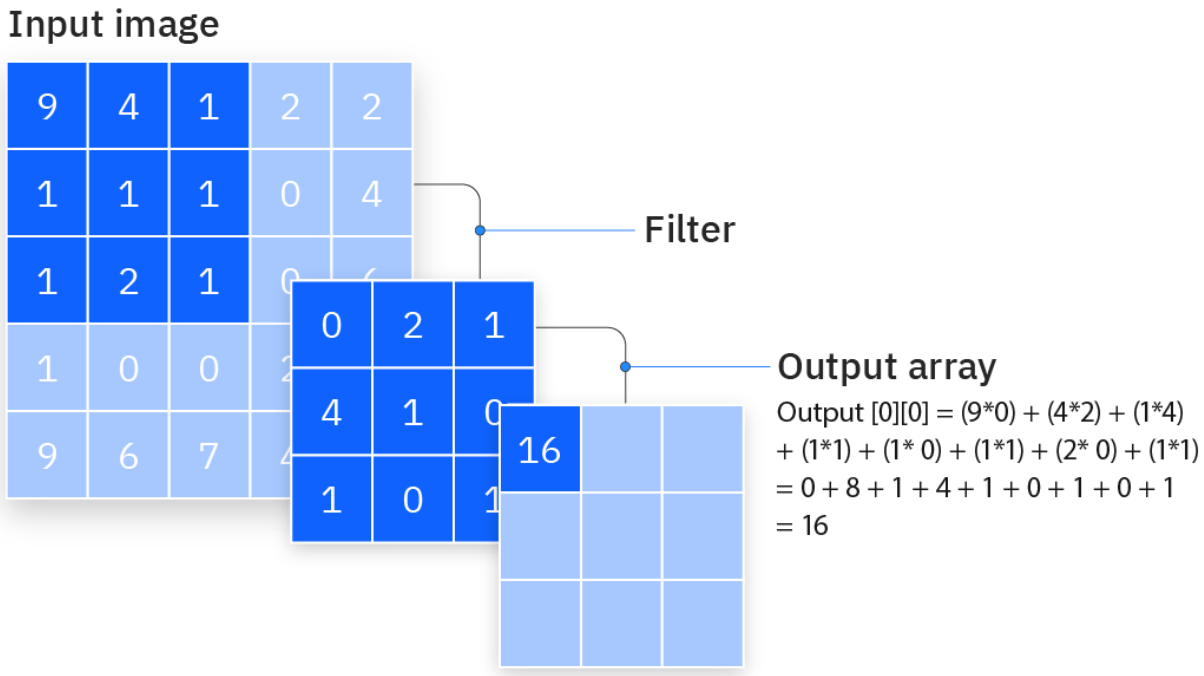
2.1.5 Convolutional Neural Network

Convolutional Neural Network (CNN) er en type nevralt nettverk som har i hovedoppgave å drive mønstergjenkjenning på bilder, og har dermed en sentral rolle i oppgaven for å kunne identifisere fartøy. CNNs muliggjør det å detektere objekter uten at nettverkene blir for krevende. Disse nettverkene gjør det mulig å utføre objektdeteksjon på en måte som er håndterbar for datamaskinens prosesseringskraft.

Et non-convolutional nevralt nettverk vil koble hver piksel i et bilde til hvert neuron i det neste laget, noe som raskt blir uoverkommelig i kompleksitet. For eksempel, et fargebilde med en oppløsning på 128 x 128 piksler ville resultere i at hvert neuron i det første skjulte laget må håndtere 49 152 vekter (128 x 128 x 3). Dette øker ikke kun beregningskompleksiteten betydelig, men slike nettverk er også mindre fleksible når det gjelder forflytning av objekter i bildet. (Mishra, 2020)

CNNs adresserer disse problemene ved å anvende et sett med filtre, vanligvis 3 x 3 eller 5x5, som beveger seg over bildet og utfører en matematisk operasjon kjent som konvolusjon. Denne prosessen har som funksjon å analysere hvordan nærliggende piksler påvirker hverandre, og kan gjenkjenne mønstre som, for eksempel, en vertikal kant. Ved å

benytte seg av konvolusjon, reduseres antall nødvendige koblinger mellom neuroner betraktelig, og nettverkets evne til å prosessere forandringer i objektposisjonen forbedres (IBM, 2023). Et eksempel på konvolusjon kan sees i Figur 2-8



Figur 2-8: En type konvolusjon. Input bildet behandles gjennom et filter og får en output på en rute med verdi 16. (IBM, 2023)

Etter at bildet har blitt ferdig behandlet av konvolusjons-filtrene, dannes det noe som kalles et «feature map», som videre behandles gjennom en aktivasjonsfunksjon, ofte benevnt som ReLU (rectified linear activation function). Funksjonen behandler data lag for lag, hvor de første lagene typisk gjenkjenner simple elementer som hjørner og kanter, mens de etterfølgende lagene identifiserer mer komplekse strukturer som fulle objekter. (Krishnamurthy, 2022)

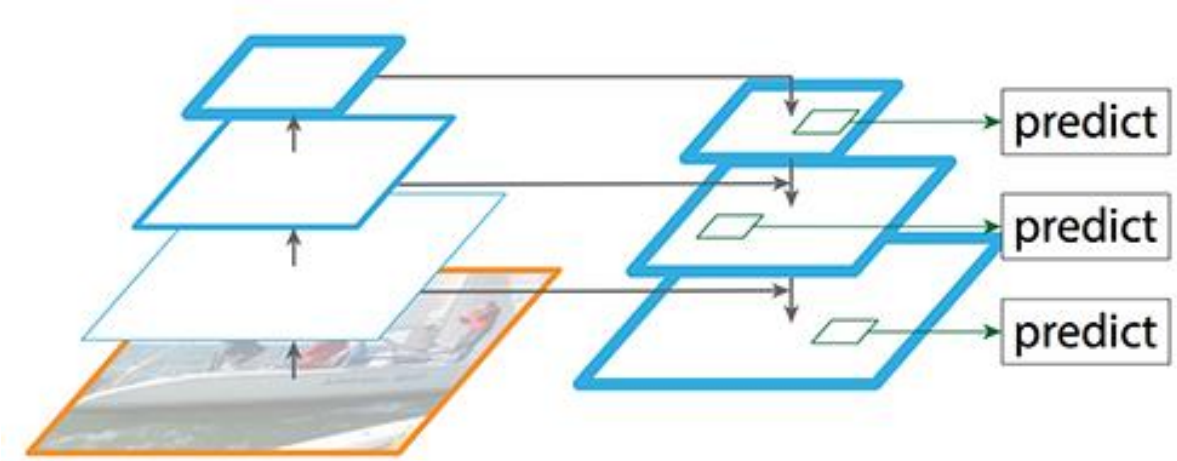
Ved å integrere pooling-lag i CNN-arkitekturen, hvor de største verdiene fra arkiveringsfunksjonen hentes ut og sendes videre til neste lag, forbedres ytterligere nettverkets evne til å klassifisere objekter i bilder.

Resultatet av dette er et nettverk som er mer effektivt og presist i mønstergjenkjenning, noe som spiller en sentral rolle for å oppnå en effektiv og nøyaktig maritim overvåkning. (Platon, 2023)

2.1.6 Feature pyramid network

Feature Pyramid Network (FPN) er en fremgangsmåte innen nevrale nettverk som forbedrer evnen til å forutsi objekter i en detekesjonsmodell. Ofte benyttes CNNs som input for FPN, noe som bidrar til å forsterke nettets prediksjonsevne. Dette oppnår FPN ved å benytte informasjonen fra flere bildeoppløsninger, noe som optimaliserer gjenkjenningen av objekter. (Hui, 2018)

En tilnærming til dette er å skape et sett av feature-maps fra ulike oppløsingsnivåer av et gitt inputbilde. Hvert enkelt feature-map inneholder viktig informasjon fra sitt respektive oppløsningsnivå, og prediksjonene gjøres baser på disse kartene. Selv om denne prosessen kan virke kompleks, er den mindre krevende enn og dirkete prosessere flere ulikt skalerte bilder samtidig. Prosessen er illustrert på figur 2-9.



Figur 2-9: Illustrasjon av oppskaleringen ved bruk av FPN (Hui, 2018)

I en FPN blir feature-maps fra forskjellige oppløsningsnivåer fusjonert gjennom en kombinasjon av vertikal og horisontal kobling. Dette innebærer en sammenfletting av feature-maps på like oppløsningsnivå (horisontalt), samt en integrasjon med feature-maps fra lavere oppløsningsnivå (vertikalt). Metodens hensikt er å forbedre nettets kapasitet til å identifisere både små og store objekter. Hvor små objekter blir mer gjenkjennelige i høyere oppløsning, mens større objekter blir lettere å identifisere i lavere oppløsning. (Hui, 2018)

Ved bruk av denne tilnærmingen kan FPN effektivt balansere behovet for nøyaktighet i forskjellige skalaer, noe som er essensielt i objektdeteksjon, spesielt i konteksten av maritim overvåkning hvor varierte objektstørrelser er en vanlig utfordring.

2.1.7 Kvalitetskriterier for maskinlæringsmodeller

Mean average precision er et mål som brukes for å evaluere eller si noe om presisjonen til en maskinlæringsmodell innenfor objektdeteksjon. Når modellen trenes regnes dette ut ved å måle modellens riktige deteksjoner opp mot det totale antallet av deteksjoner. Under denne valideringen finner algoritmen ut hva som er riktige deteksjoner, sanne positive (TP – True positive) og falske positive (FP – false positive), ved å bruke testdatasettet. Selve utregningen av mAP videre med disse variablene er gitt som forholdet mellom sanne positive (TP) og det totale antallet predikerte positive (TP + FP) (Tan R. , 2019):

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

Formel 2-10: Utregning av mean average precision (mAP)

I likhet med mAP kan man regne ut average recall. Det er en relativt enkel utregning som sier noe om maskinlæringsmodellens ytelse når det kommer til TP-raten. Dette er også referert til som modellens sensitivitet. Med TP-raten ser man på hvor mange sanne positive deteksjoner modellen har gjort, opp mot hvor mange den egentlig skulle ha gjort eller da det totale grunnlaget for positive deteksjoner. Her trekkes derfor også falske negative (FN – false negative) inn i regnestykket (Tan R. , 2019). Formelen er gitt slik:

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

Formel 2-11: Utregning av average recall (AR)

Gjennom å analysere formlene ser man at de henger tett sammen og at endringer i FP og FN påvirker begge tallene. Ved å redusere antallet falske positive (FP), oppnår modellen høy presisjon. Dette resulterer samtidig i at AR blir lavere, og det samme vil skje motsatt hvis FN blir økt (Tan R. , 2019). Når det kommer til objektdeteksjonsmodeller ønsker man som oftest høy presisjon, altså at så mange predikerte positive som mulig skal være sanne positive, noe som er meget relevant for denne oppgaven.

Videre er det også av verdi å presentere begrepet «accuracy». Dette er et mål på hvor ofte en klassifiseringsmodell gjør riktige forutsigelser. Det er en generell indikator på modellens ytelse og beregnes som forholdet mellom antall korrekte forutsigelser og det totale antall forutsigelser. Nøyaktighet er mest nyttig når klassene er balanserte, det vil si at hver klasse har omtrent samme antall eksempler (Tan R. , 2019).

2.2 Plattform for maskinlæring

Oppgaven har til nå tatt for seg teori om maskinlæring, modeller og trening av disse. Teori om konsepter som ligger bak modeller for objektdeteksjonsmodeller er også introdusert. Videre skal dette kapittelet ta for seg hvordan man får tilgang til nødvendige støtteprogrammer og funksjoner, for å bruke og trene maskinlæringsmodeller på en enhet.

I denne oppgaven brukes begrepet «plattform» i sammenheng med maskinlæring. Det refereres da til et omfattende verktøy eller rammeverk som tilbyr funksjoner og tjenester for å utvikle, trene, teste og distribuere maskinlæringsmodeller. Når man omtaler noe som en maskinlæringsplattform, inkluderer det typisk følgende aspekter:

- Verktøy og biblioteker som forenkler opprettelsen av maskinlæringsmodeller. Dette kan inkludere biblioteker for datamanipulering, statistisk analyse, og algoritmer for maskinlæring.
- Plattformen gir funksjoner for å utvikle og trene maskinlæringsmodeller, inkludert funksjoner for å velge, konfigurere, og optimalisere algoritmer. Videre medfører dette også somregel evnen til å teste og validere modellens ytelse, for å sikre at den er nøyaktig og pålitelig før den tas i bruk (Abadi, et al., 2015).
- Datahåndtering og forbehandling, evnen til å håndtere store mengder data.
- Verktøy for å rense, strukturere og forberede data for analyse er en kritisk del av en maskinlæringsplattform. (Tidemann & Elster, 2023), (Abadi, et al., 2015).

2.2.1 Tensorflow

Tensorflow er en åpen kilde plattform for maskinlæring som inneholder alle aspektene som er nevnt over. Tensorflow blir beskrevet som, «An end-to-end machine learning platform» (Tensorflow, 2023), da dette rammeverket inneholder programvare verktøy med dype nevrale nettverk for å håndtere maskinlæring flere steder i prosessen av modellutvikling.

Tensorflow er altså et grensesnitt for å tilgjengeliggjøre algoritmer innen maskinlæring. Det er også en implementasjon for å utføre disse algoritmene. Tensorflow kan kjøres på mange forskjellige systemer, fra mobile enheter som telefoner med mikroprosessorer, til store distribuerte systemer med hundrevis av maskiner og tusenvis av beregningsenheter som GPU-kort. Plattformen skal med andre ord kunne håndtere forskjellige størrelser av datasett og arbeidsbelastninger, skaleres opp eller ned etter behov, og integrere maskinlæringsmodeller i eksisterende systemer og applikasjoner. Dette gjør at tensorflow er en plattform for brukere med forskjellige ferdighetsnivåer, fra eksperter til nybegynnere i maskinlæring (Abadi, et al., 2015). Tensorflow er benyttet i denne oppgaven både til å implementere eksisterende modeller, og videretrene noen av disse.

2.2.2 Tensorflow Lite

Tensorflow Lite (TFlite) er en distribusjon som inneholder verktøy og algoritmer for å konvertere og optimalisere eksisterende tensorflow modeller for edge-enheter med begrenset kapasitet. Raspberry Pi som er benyttet i denne bacheloroppgaven er en enkeltkortdatamaskin (single-board computer), og er et eksempel på en slik edge-enhet. TFlite muliggjør derfor at man kan bruke ressursbegrensende enheter til å effektivt gjøre slutningsoppgaver (inference tasks), med krevde ferdigtrente maskinlæringsmodeller (Boesch, 2022). I diagram 2-1 er det illustrert forskjellen mellom vanlig tensorflow og tensorflow lite i interferenshastigheter, hva dette betyr er beskrevet videre i kapittelet.

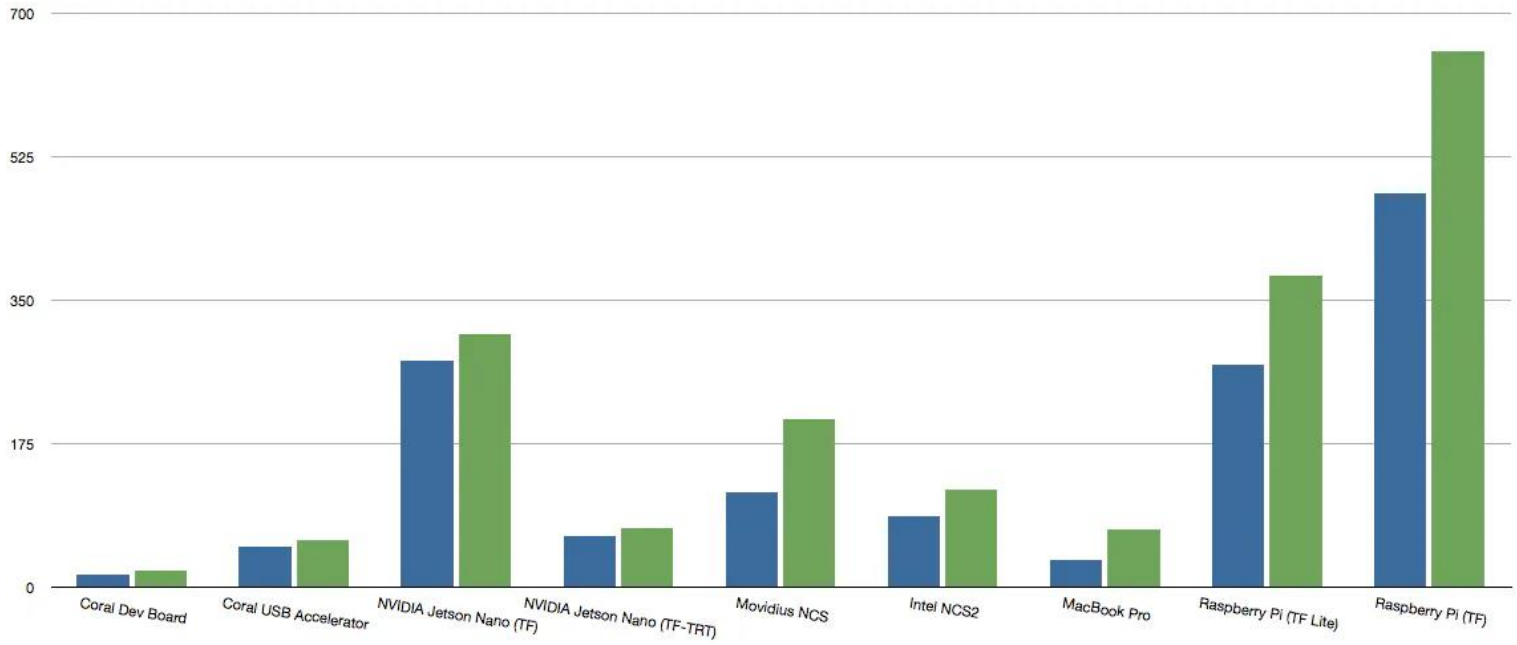


Diagram 2-1: Inferenshastigheter i millisekunder for MobileNet SSD V1 (Blå) og MobileNet SSD V2 (Grønn) på forskjellige enheter. (Allan, 2018)

Board	MobileNet v1 (ms)	MobileNet v2 (ms)
Raspberry Pi (TF Lite)	271.5	379.6
Raspberry Pi (TF)	480.3	654.0

Tabell 2-2: Tall fra diagram 2-1 for Raspberry Pi. (Allan, 2018)

Modellene som er brukt i disse testene er SSD mobilenet v1 og v2, dette er en maskinlæringsmodell som også er brukt i denne oppgaven. V2 er en modell med større nevralt nettverk enn v1, noe diagram 2-1 også bekrefter da den er tyngre for enhetene å kjøre. Interferenshastighet er noe som måles når modellene anvender det den har lært på ny data for å lage prediksjoner eller ta beslutninger, og indikerer hvor fort dette blir gjort. Diagram 2-1 og tabell 2-2 viser forskjellen i inferens mellom TFlite og Tensorflow, og man ser at de ferdigtrente modellene kjøres mer effektivt med konverteringen til TFlite. Det er likevel verdt å notere at dette går på bekostning av flere ting, som nøyaktighet og temperatur på CPU (Allan, 2018), se tabell 2-3. Dette er faktorer som må veies opp mot oppgavens mål.

Board	Peak External Temp (°C)	Peak CPU Temp (°C)
Raspberry Pi (TF Lite)	64	78
Raspberry Pi (TF)	58	74

Tabell 2-3: Maksimal ekstern temperatur og CPU temperatur i °C under inferensprosessering. (Allan, 2018)

2.3 AIS-data

> *"Det er ca. 5000 AIS-bærende fartøy i norske farvann til enhver tid. Hvert skip oppdaterer sin posisjon på AIS ca. hvert 10. Sekund" (Kystverket, 2023)*

Automatic Identification System (AIS) er et system for å automatisk å dele og motta informasjon om skipstrafikk. Denne informasjonen blir sendt ut av båter selv, og deles til forskjellige mottakere som landstasjoner, andre fartøy og relevante andre som ønsker informasjonen (International Maritime Organization, u.d.). Informasjonen er det fartøyene selv deler og inneholder data om fartøyets identitet, type skrog, posisjon, kurs, fart, navigasjonsstatus og annen sikkerhetsrelatert informasjon (UNSD_MM, 2020). AIS ble i første omgang introdusert for å lettere oppnå situasjonsforståelse og til navigasjonsinnretning, i den hensikt å lettere unngå kollisjoner. AIS omtales ofte som et antikollisjonsverktøy, men brukes også i dag til andre formål som for eksempel overvåking, og deles til interesserte på nettsider som «marinetraffic.com». AIS-enheten ombord på fartøyet (Global Navigation Satellite Systems-mottaker (GNSS) og VHF-FM transmitter) sender kontinuerlig og automatisk fartøyets posisjon, identifikasjon og andre fartøysdata. I tillegg kan den motta meldinger fra andre skip og kyststasjoner. Rapporteringsintervallet til fartøyet avhenger av en rekke faktorer som fartøyets hastighet, kurs og tilstand, som vist i tabell 2-4 (Kystverkets hovedkontor, Sjøsikkerhetsavdelingen, 2019)

Type skip	Rapport intervall
Skip til anker eller fortøyd og ikke beveger seg raskere enn 3 knop	3 min
Skip til anker eller fortøyd og beveger seg raskere enn 3 knop	10 s
Skip som har en fart mellom 0 – 14 knop	10 s
Skip som har en fart mellom 0 – 14 knop og forandrer kurs	3 1/3 s
Skip som har en fart mellom 14 – 23 knop	6 s
Skip som har en fart mellom 14 – 23 knop og forandrer kurs	2 s
Skip som har en fart større enn 23 knop	2 s
Skip som har en fart større enn 23 knop og forandrer kurs	2 s

Tabell 2-4: Rapporteringsintervall klasse A AIS (Kystverket, 2019)

AIS-meldinger består av tre hovedkomponenter: et MMSI-nummer som unikt identifiserer senderen, en meldingstype som beskriver typen informasjon som formidles, og et repetisjonsnummer som anvendes for videreformidling av meldingen. Av de 27 forskjellige meldingstypene er det spesielt tre typer, nemlig 1, 2 og 3, som er dedikert til å overføre navigasjonsinformasjon for et fartøy. Disse inkluderer data som hastighet over grunn,

kurs, posisjon og tiden for meldingens utsendelse, som demonstrert i tabell 2-5. Type 5-meldinger bærer statisk- og fartøysspesifikk informasjon, for eksempel navnet på skipet, destinasjon og lengden på skipet.

Type 1, 2 & 3
Message ID
Repeat indicator
User ID
Navigational status
Rate of turn (ROT)
Speed over ground (SOG)
Position accuracy
Longitude
Latitude
Course over ground (COG)
True heading
Time stamp
Special manoeuvre indicator

Tabell 2-5: Eksempel på meldingstype 1, 2 og 3 (Horten, Ytterstad, & Rugahiye, 2021)

AIS-stasjoner sender meldinger på et spesifikt format. Fra sine AIS-stasjoner leverer Kystverket AIS-data i standard IEC format (IEC 62320). Dette formatet er en kombinert elektronikk- og dataspesifikasjon for kommunikasjon mellom maritim teknologi. Man må derfor benytte seg av en dekoder for å få AIS-datastrømmen på et leselig format for sammenlikningsprogrammet.

For å få tilgang på AIS-dataen kreves det at man kobler seg opp på kystverket sine AIS-servere, ved bruk av en TCP/IP-socket. TCP/ IP-socketen benyttes til å sende informasjon mellom to datamaskiner, eller fra en server til mange klienter. Hvor man da benytter IP-adressen og porten på enhetene for å kommunisere. Kystverket besitter to forskjellige servere. Den ene serveren ligger åpent og er tilgjengelig for alle på IP-adresse 153.44.253.27 og port 5631. Den andre utsendelsen er mer konsis med hyppigere oppdatering, men dette er en begrenset versjon som krever en godkjent søknad til kystverket.

Den åpne serveren begrenses på databredde. Rapporteringsfrekvensen er lavere, og man har ikke tilgang på fiskefartøy under 15 meter og fritidsfartøy under 45 meter. (Kystverket, 2023)

2.4 Infrastruktur for datainnhenting

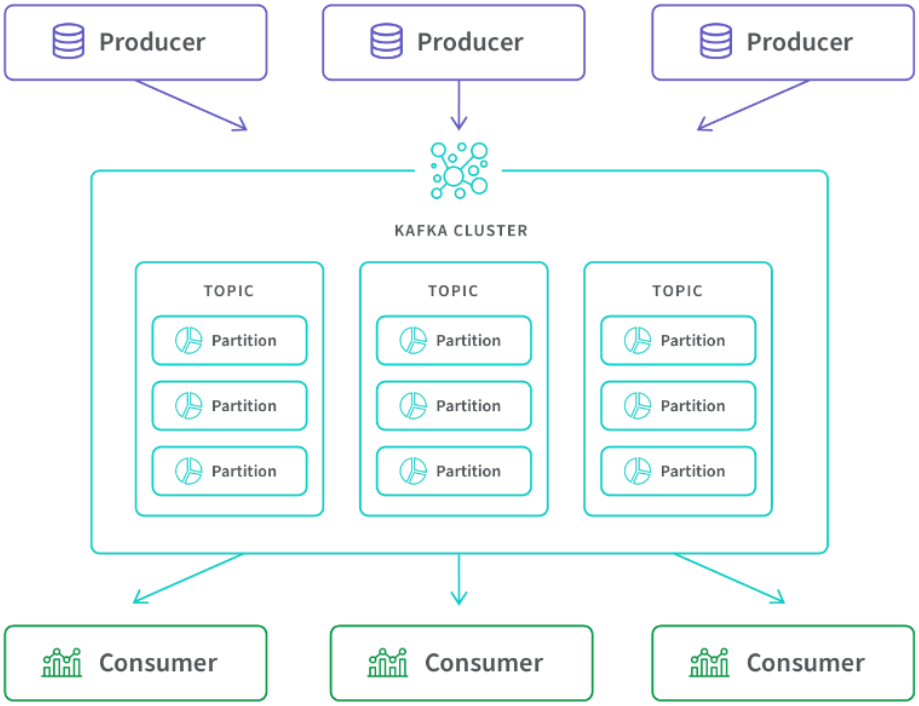
I dette delkapittelet skal det teoretiske grunnlaget bak- og programmene som er brukt for datainnhenting presenteres. AIS-dataen innhentes fra Kystverkets server, og deteksjons-dataen overføres fra RPI'en. Denne delen består av underkapitlene Apache Kafka og Flask – mikro internettrammeverk.

2.4.1 Apache Kafka

Apache Kafka kan enkelt beskrives som et publiserings- og abonnementssystem (publish/subscribe). Dette systemet fungerer som et skjøtekors mellom datakilder og -mål, og gir mulighet for realtidsanalyse av data. Når det gjelder maritim overvåking, hvor innsamling og analyse av data er kritisk, kan Kafka spille en sentral rolle i å strømlinjeforme denne prosessen. Satt på spissen er Kafka utviklet for å håndtere store datastrømmer. Kafka benyttes allerede i dag av samfunnsaktører som NAV og Statsnett for å håndtere strømmende data fra milliarder av hendelser digitalt (Jakobsen, Paramanathan, & Sommerfeldt, 2023).
Kafka har tre hovedfunksjoner:

- Publisere og abonnere på strømmer av hendelser (publish & subscribe).
- Lagre disse hendelsene over en bestemt periode (holdefunksjon).
- Behandle disse hendelsesstrømmene enten i sanntid eller ved en senere tidspunkt.

Strukturen til Kafka kan brytes ned til enkelte konsepter. En melding i Kafka er kjent som en hendelse og er organisert med en hendelsesnøkkel, en verdi (som representerer meldingsinnholdet) og et tidsstempel. Den som sender meldingen til Kafka kalles en produsent, mens den som mottar meldingen kalles en abonnent. Disse hendelsene er lagret i emner (topics), som videre kan segmenteres i partisjoner. Strukturen er illustrert i figur 2-12.



Figur 2-12: Illustrasjon av strukturen i Kafka (Qlink, 2023)

En fordel med Kafka er dens robuste disklagringsfunksjonalitet. Dette betyr at data ikke går tapt, selv om en klient kanskje ikke klarer å motta dem umiddelbart. For maritim overvåking, hvor overvåkingssystemet muligens må behandle enorme mengder data som strømmer inn, i vårt tilfelle fra en AIS-server, gir denne funksjonen en garanti for datasikkerhet. Videre kan lagringsvarigheten justeres basert på systemets behov, for eksempel ved å slette data som er eldre enn et bestemt antall timer, slik at man skalere datalagring etter behov. (Kafka, 2023)

I konteksten av et fleksibelt maritimt overvåkingssystem, kan Apache Kafka bidra til å sikre effektiv datainnsamling, lagring og behandling.

2.4.2 Flask – mikro internettrammeverk

Flask er et mikro-rammeverk for webapplikasjoner, som er skrevet i Python med en simpel kjerne i bunn som muliggjør utvidelser til sitt eget behov. Flask kan brukes til å eksempelvis sette opp en API-tjeneste og deretter skalere applikasjonen etter behov. Fordelen med å benytte seg av Flask er at det er enkelt å utvikle eksempelvis en nettside eller en API-tjeneste.

En API-tjeneste er et programmeringsgrensesnitt som tillater et program å aktivere en kode i et annet program. Dette muliggjør at en klient med mindre ressurser kan gjøre mer

komplekse oppgaver ved hjelp av et annet program. Å kjøre en nevral nettverksalgoritme kan ofte være ressurskrevende, derfor kan det være aktuelt å kjøre denne blokken med kode på en maskin som håndterer belastningen.

I konteksten av en Flask-basert applikasjon for dataoverføring, fungerer Flask som bindeleddet mellom en server og klienter. Hvor den håndterer HTTP-forespørsler og utfører filopplasting og -nedlasting gjennom definerte endepunkter. Flask håndterer disse oppgavene med en kombinasjon av Python-kode og HTTP-protokollen, som sikrer enkel dataoverføring mellom enheter i et nettverk. (Python Basics, 2023)

Ved å eksempelvis sette opp en flask-server på en enhet, slik at andre enheter kan overføre data til serveren, kan dette bidra til et fleksibelt system for maritim overvåkning.

2.5 SQLite

SQLite er en lettvektig, serverløs SQL-database som kan brukes til lagring og håndtering av data. Den støtter fullverdig SQL-syntaks, noe som gir den mulighet til å utføre intrikate databasemanipulasjonsoppgaver og forespørsler. (SQLite, 2022)
Fordelen ved å bruke SQLite er at den er enkel å ta i bruk. Det er også åpen kildekode programvare som blir brukt i svært mange applikasjoner, noe som gjør det enkelt å finne dokumentasjon på implementering og håndtering av databasen.
Det er også utviklet et visualiseringsverktøy for SQLite ved navn DB Browser for SQLite (DB4S). DB4S kan brukes til å analysere, navigere, designe og redigere filer i databasen. (DB Browser for SQLite, 2021) Dette øker brukeranvendeligheten til databasen.

2.6 Distribuerte systemer (Distributed Systems)

Distribuerte systemer er datasystemer som består av flere uavhengige komponenter, hvor alle de forskjellige komponentene i systemet arbeider for å oppnå et felles mål. Det som er spesielt med slike systemer er at komponentene faktisk er fysisk adskilt, og videre kan de også være lokalisert på forskjellige geografiske steder. Man er derfor avhengig av at disse komponentene kommuniserer og koordinerer med hverandre over et nettverk (Soulaimaneyh, 2023). Dette er altså en beskrivelse av et system som kan se ut som et enkelt system for brukeren, men som egentlig er satt sammen av flere komponenter eller «delsystemer». Hovedforskjellen fra konvensjonelle datasystemer er altså at distribuerte systemer må kommunisere og koordinere mellom komponententene sine ved å sende meldinger til hverandre over et nettverk (Soulaimaneyh, 2023).

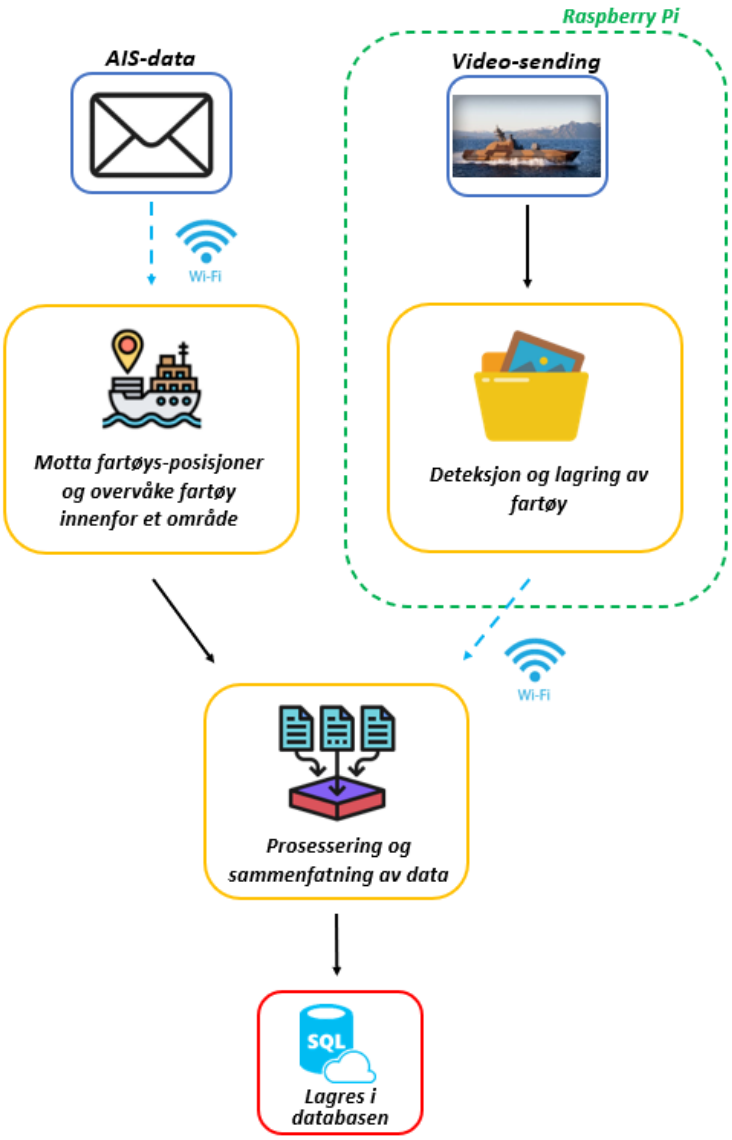
3 Implementering

I teoridelen har det nå blitt introdusert et teoretisk grunnlag for å forstå de overordnede prinsippene innenfor maskinlæring, og hvordan rammeverket rundt dette fungerer. Videre har teorien også tatt for seg AIS systemet, og hvordan denne dataen kan samles inn. Prosjektet skal benytte objektdeteksjon med et overvåkningskamera, i kombinasjon med en enhet for innhenting av AIS-meldinger som også brukes til sammenligning og sammenfatning i en database. Ved å benytte data fra åpen-kilde deteksjonsmodeller skal kameraet være i stand til å kontinuerlig detektere og lagre passerende fartøy. For å kunne drive maritim overvåking og datainnhenting er det flere systemer og komponenter som samhandler. Systemet er derfor basert på konseptet bak distribuerte systemer, da systemet i seg selv belager seg på samhandling mellom komponenter som er fysisk adskilt.

En av komponentene i systemet er en Raspberry Pi 4 (RPI), sammen med et Raspberry Pi HQ Kamera. Dette er enheten hvor det drives objektdeteksjon og videresending av deteksjoner på. RPI'en var allerede tilgjengelig på skolen, HQ kameraet ble kjøpt inn og objektdeteksjonsmodellene er hentet gjennom åpenkilde kode på nettet.

Den andre komponenten av systemet er en bærbar computer som henter inn AIS-meldinger. Denne delen av systemet er inspirert av bacheloroppgaven til Horten, Ytterstad og Rugahiye fra 2021 (Horten, Ytterstad, & Rugahiye, 2021). Videre benyttes en Flask-server for å innhente data fra Raspberry Pi'en. Deretter sammenlikner et program på computeren kameradeteksjonene med relevant AIS-data, og lager en database hvor deteksjonene lagres.

Figur 3-1 illustrer hvordan systemet er bygget opp, hvor alle prosesser som foregår på Raspberry Pi'en er avgrenset av den stiplede boksen. Resterende prosesser foregår på den bærbare computeren, og figuren illustrerer at det er et distribuert system.



Figur 3-1: Illustrasjon av overordnet system

3.1 Kameraoppsett

Et av systemkravene var at oppsettet skulle være mobilt og kunne plasseres der bruker ønsker å ha det. Dette er en faktor som har påvirket implementeringen av systemet, spesielt innenfor valg av maskinvare og programmer. Det var mye å ta hensyn til når det kommer til mobilitet, alt fra størrelse og vekt, til installasjonstid på ønsket sted og kommunikasjon videre herifra. Som en fortsettelse på bacheloren til Hollup og Røsand ble det naturlig å velge tilnærmet lik maskinvare også i dette prosjektet (Hollup & Røsand, 2022). Ved å velge en Raspberry Pi (RPI) sammen med et tilhørende Picamera som hovedkomponenter for objektdeteksjon, kunne denne delen av systemet oppnå mobilitet uten altfor stor kompleksitet. Ved å installere nødvendig programvare og maskinlæringsmodell på

RPI'en, kunne denne komponenten i det overordnete systemet fungere fullstendig alene for deteksjon og lagring av deteksjonsdata. I de følgende avsnittene gis en mer detaljert beskrivelse av delkomponentene og deres konfigurasjon.

3.1.1 Raspberry Pi

I denne oppgaven ble det brukt en Raspberry Pi 4 som maskinvare for objektdeteksjon. Som nevnt tidligere i kapittel 2.2.2 er dette en enkeltkorts-datamaskin, og inngår i det man omtaler som edge-enheter. Den har begrenset kapasitet, noe som har sine fordeler og ulemper. Den åpenbare ulempen er at maskinen ikke kan belastes med for mange og store programmer, men dette resulterer derimot i at maskinen er liten og meget håndterbar. I dette prosjektet er RPI'en koblet med strømkontakt direkte fra veggen, for å gjøre den til en ren mobilenhet vil det kunne legges til en batteripakke som gjør enheten uavhengig av strømuttak, men dette ble ikke prioritert. En mulig løsning med batteri er demonstrert andre steder tidligere, som for eksempel på dronene fra bacheloroppgaven til Hollup og Røsand fra 2022 (Hollup & Røsand, 2022).

Figur 3-2: Raspberry Pi 4 (Farnell, 2023)

Maskinlæringsprogrammer for objektdeteksjon kan være krevende å kjøre, og prosessorkraft utgjør mye av hvor godt en modell kjører. For å gi RPI'en best mulig forutsetning til å detektere og videresende lagret informasjon, ble dette de eneste oppgavene denne delkomponenten skulle gjøre. Dette medførte riktignok også at enheten måtte integreres i et nettverk for å sende informasjon til andre enheter, som i dette prosjektet er en bærbar PC. Ved å bruke internett og en Flask-server, deles informasjonen mellom enhetene. Systemet med denne løsningen er fortsatt meget mobilt, dog setter det noen begrensninger da systemet må være innenfor internettrekkevidde. Hvis RPI'en ikke har internettilgang, vil

den fortsatt detektere og lagre informasjonen lokalt da denne prosessen ikke krever internett. Det vil si at disse dataene kan sammenlignes for identifikasjon med AIS i etterkant, da AIS-meldingene er lagret på PC'en. Slik systemet er bygd opp nå, kan data slettes etter de er sendt videre lokalt på RPI'en. Uten internett kan man ikke slette dataen, noe som kan bli utfordrende i lengden da RPI'en har begrenset med lagringsplass.

3.1.2 Kamera og linse

Konseptet for prosjektet går ut på å overvåke og lage datapakker som kan brukes videre til spesifikke maskinlæringsmodeller i ønskede områder. Det er derfor vesentlig at kameraet som brukes med RPI'en, er et godt nok kamera til å se detaljer av båtene på avstander som det er naturlig at fartøy har til land i indre kyststrøk. Etter å ha analysert resultater fra tidligere oppgaver (Hollup & Røsand, 2022) sammen med egne tester, ble det konkludert med at det tilgjengelige «Camera V2» fra Raspberry pi ikke tilfredsstilte kvalitetskravet på de avstandene dette prosjektet krever. Grunnen til dette var at bildene ikke hadde god nok oppløsning. Det ble derfor kjøpt inn et nytt og kraftigere HQ PiCamera med tilhørende linse til bruk som optisk sensor. HQ Picamera (High Quality Raspberry Pi Camera) er et Sony-kamera fra Raspberry Pi, og er derfor enkelt å koble opp og bruke med RPI'en. Dette kameraet har oppløsning opp til 12,3 megapiksler og er kompatibelt med flere typer linser. Kameraet har omtrent 50 % større område per piksel i motsetning til den gamle kameramodulen v2. Dette bidrar til forbedret ytelse under dårlig opplyste forhold, som er relevant for bruken av kameraet i denne oppgaven (Raspberry Pi, 2023).

Figur 3-3: HQ Picamera (Raspberry Pi, 2023)

Linsen som ble brukt er et 16mm teleobjektiv for Raspberry Pi HQ-kamera. Denne linsen kan justeres på flere måter manuelt, deriblant fokus og lys. Ved bruk av denne linsen kan systemet ta bilder med en oppløsning opptil 10 megapiksler.

Figur 3-4: 16mm Telephoto Lens for Raspberry Pi HQ Camera. (Raspberry Pi, 2023)

3.1.3 Kamerahus og komplett oppsett

For å sette sammen RPI'en og HQ kamera til et komplett og funksjonelt oppsett ble det 3D-printet et kamerahus (Ruiz Brothers, 2020). Videre ble RPI'en og kamera skrudd fast med skruer sammen med 3D-printen. Til slutt ble det komplette kameraet satt på et tripodkamerastativ. Se vedlegg A for instruksjoner.

Figur 3-5 og Figur 3-6: Komplett kameraoppsett

3.2 Objektdeteksjon på Raspberry Pi

Objektdeteksjon kan gjøres på mange måter, med forskjellig utstyr. Dette kan være store maskinlæringsmodeller som setter krav til prosessorkraft på maskinvaren, men som beskrevet i kapittel 2.2.2 kan det fint gjøres på mindre enheter med riktig tilpasning. Denne delen av oppgaven skal ta for seg implementasjonen og oppbygning av programmene på RPI'en.

3.2.1 Operativsystem og plattform for maskinlæring

For å hente kildekode og videreutvikle denne til prosjektets formål, trengte vi et operativsystem på RPI'en. Valget av operativsystem endte på Raspberry Pi OS 10 Bullseye (Tidligere kalt Rasbian). Som det gamle navnet Rasbian hinter til er Raspberry Pi OS en Debian basert Linux-distribusjon som er spesialdesignet for Raspberry Pi maskinvaren. Dette er det offisielt støttede operativsystemet fra Raspberry Pi, og inkluderer derfor en rekke forhåndsinstallerte programmer og verktøy som er tilpasset for bruk på RPI enheter (Raspberry Pi, 2023).

Med Raspberry Pi OS til å håndtere maskinvaren og som grunnlag til å kjøre programvare, trengtes det en plattform til å implementere en maskinlæringsmodell. Valget ble Tensorflow, og enda mer i detalj Tensorflow Lite (TFlite). Det finnes mange maskinlæringsmodeller, og TFlite er en plattform som er kompatibel med flere av disse. Dette gjorde at valgmulighetene rundt valg av objektdeteksjonsmodell var større, og åpnet opp for å kunne teste forskjellige modeller opp mot hverandre. Gjennom bruk av terminalvindu ble de originale tensorflow bibliotekene installert på RPI'en (Tensorflow, 2023), se vedlegg E.

3.2.2 Maskinlæringsmodell

Etter som Tensorflow Lite er en plattform som kan brukes sammen med mange forskjellige modeller, endte valget for modell til testing og gjennomføring av prosjektet på fire forskjellige modeller. De første modellene som ble valgt var EffcientDet-Lite modellene (Tan, Pang, & Le, 2020). Dette er en serie av objektdeteksjonsmodeller som er skalert i forskjellige størrelser fra 0 til 7, og krever mer prosessorkraft for hver modell oppover mot 7. Presisjonen eller deteksjonene blir derimot mer nøyaktig jo kraftigere modellen blir, da modellene er trent med et større nevralt nettverk (Abadi, et al., 2015). Hollup og Røsand sin oppgave fra 2022 testet og kom frem til å bare ta med versjon 0 og 1 videre,

da disse var de som fungerte best på Raspberry Pi 4 (Hollup & Røsand, 2022). I denne oppgaven er det også kun valgt å gå videre med de minst krevende modellene, henholdsvis 0, 1 og 2, da vi benytter oss av en identisk RPI som i oppgaven fra 2021.

Det ble også valgt å ta med en modell til, ssd_mobilenet_v2 FPN-lite. Dette ble gjort av den grunn at dette var en modell som fort og enkelt kunne trenes ved hjelp av transfer learning. Denne modellen ble implementert på RPI'en gjennom en modifisert kildekode av Tensorflow lite, men som i prinsippet fungerer identisk (Juras, Tielens, Trax, & Hong, 2022). Modellene er presentert i tabell 3-1, med data hentet fra Kaggle og Tensorflow (Kaggle, 2020) og (Tensorflow, 2023).

MODEL	STØRRELSE(MB)	FORSINKELSE(ms)	mAP (På COCO 2017 validerings datasett)
EfficientDet-Lite 0	4.4	37	25.69%
EfficientDet-Lite 1	5.8	49	30.55%
EfficientDet-Lite 2	7.2	69	33.97%
ssd_mobilenet_v2 FPN-lite	9.3	-	22.2%

Tabell 3-1: Oversikt over testede modeller. (Kaggle, 2020) og (Tensorflow, 2023)

Alle fire modellene er trent på samme datasett COCO 2017. Dette er et datasett som inneholder over 200 000 treningsbilder og har 80 klasser av forskjellige objekter (Kaggle, 2020). En av disse klassene er objektet «Boat». Prosjektet vårt krever i prinsippet derfor ikke noe trening for å fungere, men det ble allikevel valgt å ta med ssd_mobilenet_v2 FPN-lite modellen som kunne trenes av to forskjellige grunner. Hovedgrunnen var at dette var en fin måte å validere at de innsamlede bildene kameraet tar, faktisk er gode nok til å trene en modell, som til syvende og sist er systemets faktiske oppgave. Den andre grunnen var at hvis modellene som ikke var videretrent ikke hadde klart å detektere båter som ønsket, var det nødvendig å trene en modell til dette for å teste konseptet.

3.2.3 Trening

Det ble i dette prosjektet utført transfer learning av ssd_mobilenet_v2 FPN-lite modellen, det vil si at de allerede trente vektene i nettverket ble trent videre med et mindre datasett. Bildene i datasettet er manuelt samlet inn gjennom RPI'en sammen med det tilhørende HQ kameraet. Bildene som ble samlet inn ble tatt med relativt lik vinkel på vannet og

fartøyene, men det var varierende grad av støy i bildene. Med det menes at noen bilder inneholdt tilnærmet kun vannet og båten, mens andre bilder inneholdt forskjellige objekter og varierende lysforhold. Det ble totalt brukt 211 bilder, og for å gjøre mappen med bilder om til et «datasett» ble det brukt et program som heter labelImg (se vedlegg C). Dette er et program som kan lastes ned og som brukes til å annotere hvor på bildene objekter man søker er, dette blir gjort ved å markere objektet med en boks. Det blir da opprettet en XML-fil tilhørende hvert bilde som inneholder annoteringsdata, denne dataen beskriver pikselkoordinater for hvor i bildet objektet befinner seg (PyPi, 2021). Etter denne prosessen inneholdt mappen med data 422 elementer, 211 bilder med hver sin korresponderende XML-fil. Se figur 3-7 for eksempelbilder som ble brukt til trening, disse bildene ble tatt manuelt, men med samme oppsett som modellen i prosjektet kjøres på.

Figur 3-7: Eksempelbilder fra datasettet som modellen er videretrent på

For å gjøre selve treningen ble det brukt en «google colab notebook» som er utviklet av de samme som modifiserte kilde-koden til tensorflow som forklart tidligere (Juras, TensorFlow Lite Object Detection API in Colab, 2023). Google colab er en virtuell maskin i nettleseren som er komplett med et Linux-operativsystem, filsystem, Python-miljø, og kanskje det viktigste er at man har tilgang på GPU. Denne leveres også med de fleste bakgrunnskrav for TensorFlow forhåndsinstallert (Juras, TensorFlow Lite Object Detection API in Colab, 2023). Notatblokken er bygget opp med tensorflow sine offisielle treningsbiblioteker.

Før treningen ble datasettet delt inn i trenings-, test- og valideringsbilder i notatblokken. Denne fordelingen ble gjort helt tilfeldig mellom alle bildene i settet, men det ble fordelt henholdsvis 80% til trening, 10% til test og 10% til validering. For treningen ble det også valgt noen andre parametere. Antall steg eller iterasjoner (Number of iteratios/steps) er hvor mange ganger modellen skal oppdatere og justere sine vekter for å optimalisere seg, en «batch» med bilder blir gjennomgått i hver iterasjon. En batch (Batch size) er en liten gruppering av bildene, og i denne treningen er en batch valgt til å være 16. Antall epoker sier noe hvor mange ganger alle bildene i treningsettet, altså totalt 168 stk, har blitt gått igjennom under treningen. Epoker regnes ut ifra batch størrelsen og antall iterasjoner/steg, dette er et viktig tall som sier oss hvor mange ganger alle bildene faktisk blitt gått igjennom og kan justeres for å optimalisere modellen (Abadi, et al., 2015). Utregningen og formel for epoker, samt en oversikt over parameterne for trening vises i formel 3-8 og tabell 3-2.

$$Epoker = \frac{Antall\ iterasjoner}{Antall\ batcher\ per\ epoke} = \frac{40\ 000}{\frac{168}{16}} = 3\ 810$$

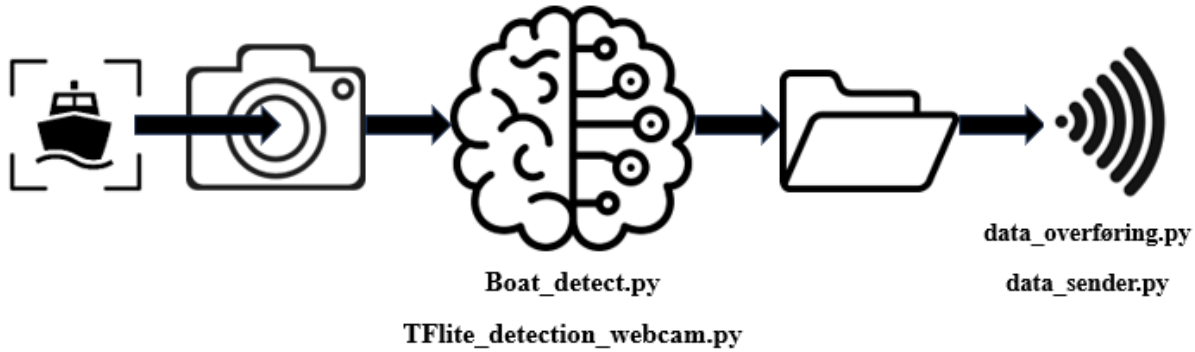
Formel 3-8: Utregning av antall epoker

Model	ssd_mobilenet_v2 FPN-lite
Number of iterations/steps	40 000
Batch size	16
Epochs	3 810
Treningsbilder	168 stk (80%)
Testbilder	22 stk (10%)
Valideringsbilder	21 stk (10%)

Tabell 3-2: Tabell med oversikt over parameterne brukt i trening

3.2.4 Programmer og filstruktur på RPI

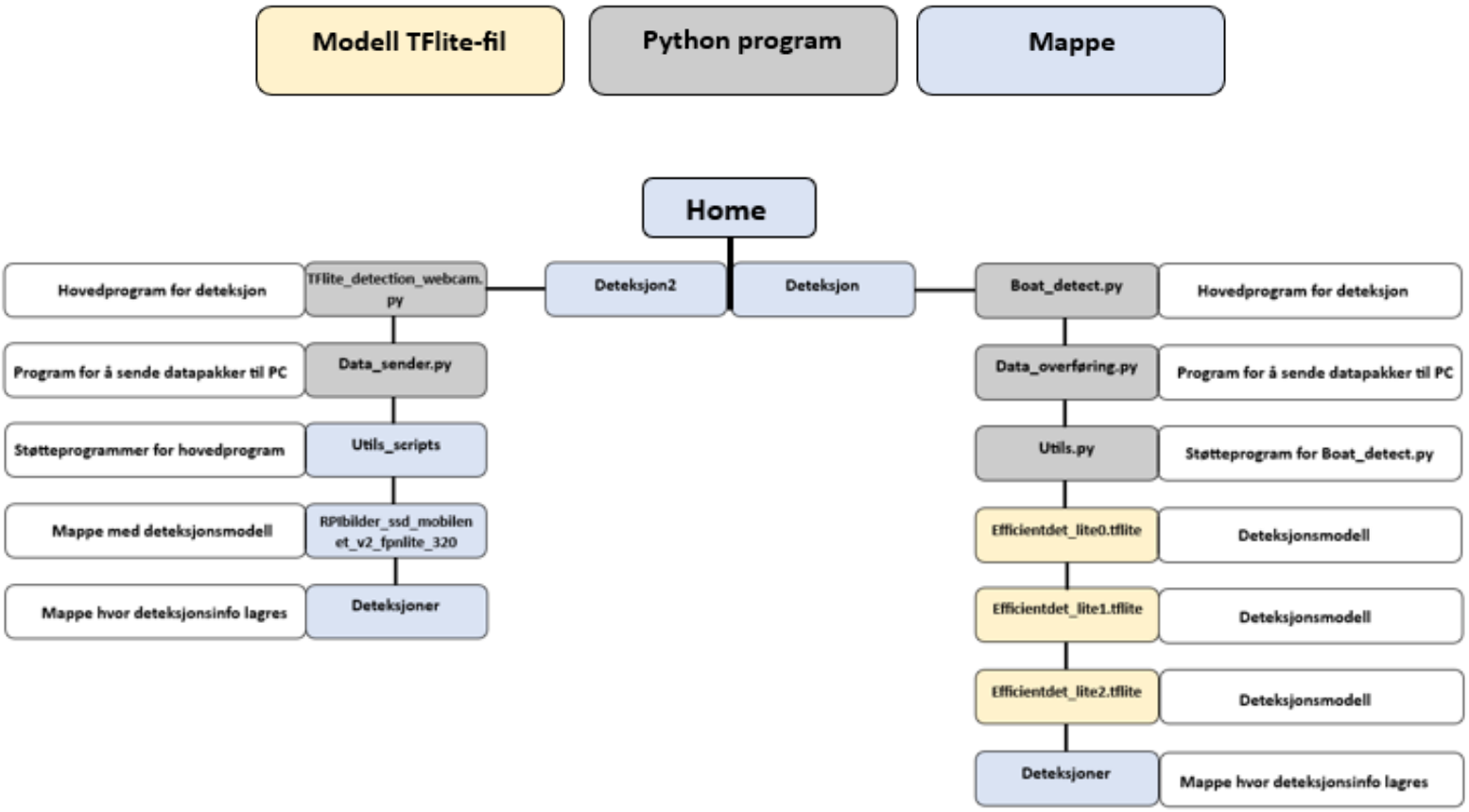
Til nå har implementeringsdelen av oppgaven tatt for seg maskinvaren og det overordnete systemet som er valgt å bruke i prosjektet. Videre i dette underkapittelet skal oppgaven ta for seg programmene som kjøres, og filstrukturen på RPI'en. Figur 3-10 viser RPI-komponenten sin overordnede funksjonalitet.



Figur 3-9: Oversikt over funksjonalitet for objektdeteksjon på Raspberry Pi

Det er to hovedprogrammer som kan kjøres, det ene programmet er kopiert direkte fra kildekoden til Tensorflow (Tensorflow, 2021), og det andre programmet er kopiert fra en

GitHub-bruker som videreutviklet programmet til å fungere med modellen vi fikk trent (Juras, Tielens, Trax, & Hong, 2022). Begge hovedprogrammene er så og si identiske, og begge har blitt modifisert i dette prosjektet til å lagre bilder og deteksjonsdata. Hovedprogrammene heter henholdsvis boat_detect.py og TFlite_detection_webcam.py. Disse er delt inn i overordnede mapper som heter deteksjon og deteksjon2, se figur 3-9 under for den fullstendige RPI filstrukturen.



Figur 3-10: RPI filstruktur

Begge hovedprogrammene lagrer deteksjoner i sine respektive «deteksjoner»-mappe. Inne i disse mappene lages det en mappe for hver deteksjon, i denne mappen lagres det bilder av fartøyene sammen med en overordnet tekstfil som inneholder data om alle bildene. Det er to kriterier for at en deteksjon skal lagres, det må være et objekt av klassen «boat» og sannsynligheten på deteksjonen må være over 50%. Grunnlaget for 50% er at det er viktig å få tatt bilder av så mange fartøy som mulig, samtidig som modellen er relativt sikker på at det faktisk er en båt i bildet. Som nevnt tidligere er det også viktig å få tatt så detaljerte bilder som mulig for å lage gode datasett, det er derfor essensielt at

kameraet plasseres på en slik måte at det er mulig for kameraet å fokusere på så få båter som mulig samtidig. Dette kan gjerne være på et sted som båter bare passerer, men hvor det fortsatt er mye trafikk, for eksempel ved smalere tilløp til store havner eller byer. Da får man færre båter i kamerabildet samtidig, og man får båtene nærme kameraet som gir gode forutsetninger for detaljerte bilder.

Dataen som lagres i tekstfilen består av tidspunkt, hvilken klasse som har blitt detektert, sannsynligheten for deteksjon, hvilket bilde dataen tilhører, boks-koordinater (rammen rundt objektet i pikselkoordinater) og de ekte koordinatene til deteksjonen i lengde- og breddegrad. All dataen utenom bredde- og lengdegrad klarer koden å hente ut ved hjelp av hovedprogrammene, fordi dataen inngår i objektdeteksjons-prosessen. Tidspunktet er et tidsstempel fra når deteksjonene ble gjort, og klassen er en variabel fra klassebiblioteket til algoritmen, men denne vil alltid være «boat» da det er den eneste klassen som vil føre til lagring av deteksjon. Når bilder lagres får de navn med tidsstempel, og dette blir inkludert i tekstfilen for å kunne navigere frem til hvilket bilde som hører til dataen. Boks-koordinatene er pikselkoordinatene for rammen algoritmen lager rundt objektet, og koordinatene referer henholdsvis til hjørnet øverst venstre og hjørnet nederst høyre. Disse boks-koordinatene er viktig for å kunne lage et datasett, da de forteller oss hvor i bildet objektet er og fungerer som annoteringsinformasjon.

Koordinatene i lengde- og breddegrad ble laget gjennom en manuell prosess. Med det menes at når kamera ble satt opp, så ble koordinatene til sentrum i kamerabildet funnet ved hjelp av sammenligning med kart. Alle deteksjoner som lagres får disse koordinatene. For å få tatt gode og detaljerte bilder er man avhengig av å komme relativt nær båtene, og unngå at kamrabildet dekker et for stort område. Ved at kameraet ikke dekker et for stort område er det ikke noe problem at deteksjonene får samme koordinater, da de er tilnærmet identiske med de faktiske koordinatene. Videre blir disse koordinatene og AIS dataen sammenlignet med en +/- buffer som gjør at dette ikke er noe problem. En detaljert forklaring på sammenligningen kommer senere i implementeringen.

For å oppsummere er det to mulige måter å kjøre objektdeteksjon på RPI'en. Metode en er å bruke «deteksjon» mappen, og kjøre programmene boat_detect.py og data_overføring.py. Her kan du velge mellom å kjøre standardmodellene efficientDet_lite 0, 1 og 2. Den andre metoden er å bruke «deteksjon2» mappen, og kjøre programmene TFlite_detection_webcam.py og data_sender.py. Her vil du kunne kjøre den trente modellen. Begge metodene vil detektere båter, lagre deteksjonen og sende deteksjonsdataen videre.

En detaljert beskrivelse for hvordan man kjører programmene ligger i vedlegg E, og funksjonalitet er beskrevet i figur 3-10.

3.3 AIS-data

Dette delkapittelet belyser hvorfor vi har valgt å benytte oss av AIS som datakilde, samt hvordan dataen prosesseres og hvilke verktøy som benyttes for å oppnå dette.

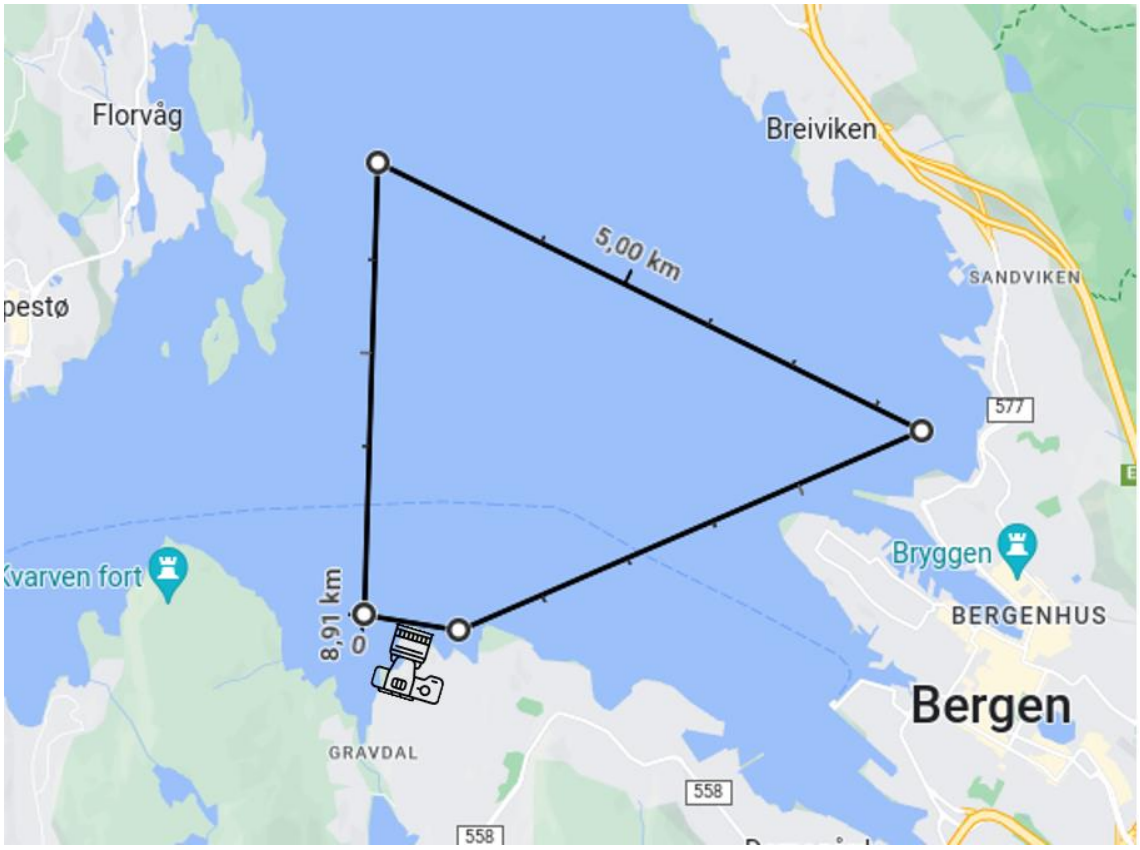
3.3.1 Valg av AIS som datakilde

Det første valget av en datakilde som kan identifisere fartøy falt på AIS. AIS benyttes av 5000 fartøy i norske farvann til enhver tid (Kystverket, 2023). Dette betyr at teknologien er godt testet og i omfattende bruk. AIS gir oss tilgang på sanntidsinformasjon om fartøy, hvor dataen er relativt pålitelig og mottas langs hele norskekysten. Kystverket leverer AIS-data fra hele norskekysten gratis til alle som trenger dette, ved bruk av en åpen server som man kobles til gjennom TCP/IP.

En av svakhetene ved bruk av AIS-data, er at fartøy kan slå av AIS'en og dermed ikke sende ut data, samt at mindre fartøy ikke er pålagt å ha AIS til å begynne med. I tillegg kan hackere benytte seg av spoofing eller jamming, som betyr at man kan få AIS-systemet til å tro at det er et fartøy et sted det ikke er, utgi feilaktig informasjon om et fartøy, eller får deg til å miste AIS-tracken til fartøy. Derfor er det naturlig å benytte seg av en datakilde som fanger opp fartøy uten, eller med falsk AIS-utsendelse.

3.3.2 Avgrensning av AIS-data

Som nevnt tidligere i oppgaven, dekker kameraet til RPI'en alltid bare et avgrenset område om gangen. Dermed faller det også naturlig å avgrense AIS-dataen til området RPI'en overvåker, i den hensikt å effektivisere systemet ved å unngå behandling av irrelevant data. Avgrensningen av området er illustrert på figur 3-8.



Figur 3-11: Illustrasjon av avgrensning for AIS-datainnhenting. Sammenlagt avstand til alle sider tilsvarer 8,91km, hvor hver gradering er 500m (Google, 2023)

Ved å benytte seg av faste holdepunkter i kamerabildet til RPI'en, samt AIS-posisjoner til passerende fartøy, klarte vi å lokalisere FOV'en (Field of View) til kameraet. Ved å videre benytte seg av objektdeteksjonsprogrammet til RPI'en, klarer vi å identifisere maksimal avstand for deteksjoner, og dette er siste parametere som kreves for å oppnå den fullstendige avgrensningen.

3.3.3 Prosessering av AIS-data

En AIS-melding sendes ved hjelp av en spesifikk kodet protokoll, og dermed må vi integrere en dekoder for å gjøre meldingen leselig. Vi løste dette ved å lage et Python-script for å dekode meldingene. Valget av programmeringsspråk falt naturlig på Python da det finnes flere nyttige biblioteker for prosessering og filtrering av AIS-data. Biblioteker som «ais» og «matplotlib» muliggjør dekodingen og filtrering av relevant informasjon fra AIS-datastrømmen fra kystverkets server.

Rådata	!AIVDM, 1, 1, , B, 13PRrB0000OvbS@NhA9=oPbr0u, 0*58
Dekodet	205743000, 1, 1, 5.1705, 60.80361, 0.0, 1, 218.699996948242, 313,0,9719305, ONJH, koksijde, 80, 21, 22, 30, NOMON, 7.400000095367432
Dekodet og Manipulert	205743000, 1, At anchor, 5.1705, 60.80361, 0.0, 1, 218.699996948242, 313,0,9719305, ONJH, koksijde, 80, 21, 22, 30, NOMON, 7.400000095367432, 2023-10-20_20-32-45
Filtrert	ID: 1, MMSI: 205743000, Y: 60.804, X: 5.170, TIMESTAMP: 2023-10-20_20-32

Tabell 3-3: Eksempel på en kodet, dekodet, dekodet og manipulert, og filtrert AIS melding.

Når dataen er dekodet må meldingen manipuleres og filtreres for å kunne hente ut den relevante informasjonen. Deretter lagres dataen slik at det kan benyttes i videre kode, og som en begivenhetsstrøm til Apache Kafka. Tabell 3-3 viser prosessen fra vi mottar dataen, frem til den er ferdig prosessert og klar til å lagres. Som man kan se i tabellen, har det blitt konvertert tallkoder til forståelige setninger. Eksempelvis har tallverdien (1) for fartøysstatus, blitt konvertert til setningen «At anchor». Dette er gjennomført ved å benytte seg av en konverteringsfunksjon basert på et oppslagsverk for de standardene som AIS-meldingsformatet bruker. I tillegg er det lagt til dato og tidspunkt for når AIS-meldingen ble sendt.

For å kunne benytte den dekodede og manipulerte AIS-meldingen til sammenlikning og lagring, må den først filtreres. Hvor vi da ekstraherer følgende kolonner:

ID, MMSI, Y_POS, X_POS, TIMESTAMP

Årsaken til den kraftige filtreringen er at vi kun henter ut data som er essensielt for at systemet skal fungere som ønsket, i den hensikt å øke påliteligheten til systemet. De ekstraherte kolonnene brukes til sammenlikning med deteksjonsdataen, samt muliggjør for identifisering av fartøy som er laget i databasen. Som nevnt i teorien om AIS finnes det flere meldingstyper, og flere av typene inneholder ulik informasjon. Det ble dermed bestemt å forholde seg til den dataen som var mest konsekvent og hyppigst oppdatert, hvor

det derfor ble meldingstypene 1, 2 og 3, grunnet at alle inneholder den samme dataen. Kolonnen ovenfor består dermed av følgende data:

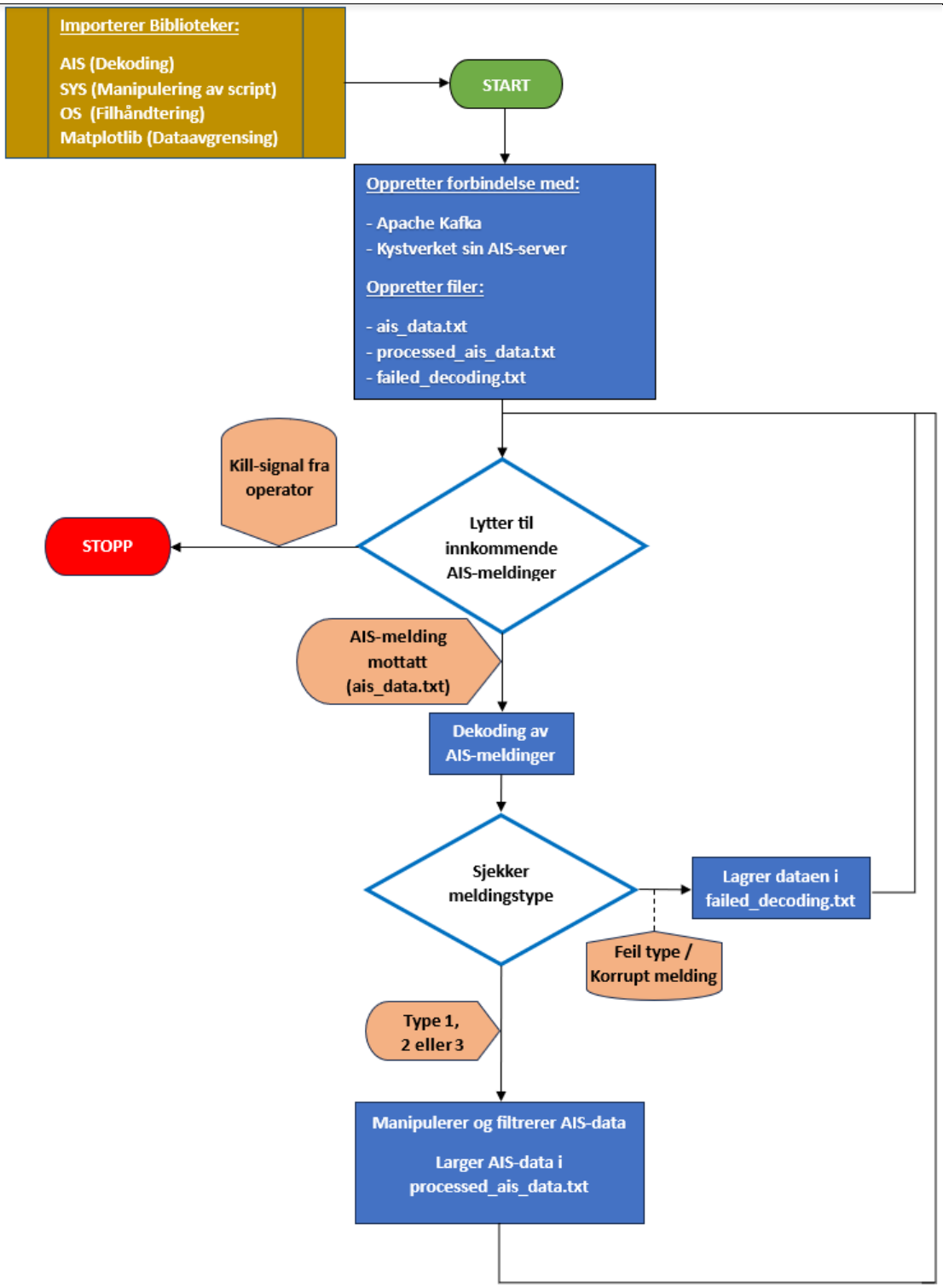
ID – Gir oss en oversikt over meldingstypene som er mottatt.

MMSI – Er et unikt nummer alle fartøy som benytter AIS besitter. Siden nummeret er unikt, muliggjør dette identifisering av deteksjonen som er lagret i databasen.

Y_POS & X_POS – Oppgir fartøyets posisjon i lengde og breddegrad.

TIMESTAMP – Forteller oss når AIS-meldingen blir innsendt.

Python-programmet som utfører all prosessering av AIS-data er lagt ved i sin helhet i vedlegg D. For å få et overordnet blikk over virkemåten til programmet kan en studere flytdiagrammet i figur 3-11.



Figur 3-12: Flytdiagram for AIS-program

For at AIS-dataen enkelt skal kunne benyttes som sanntidsdata, benyttes Apache Kafka. Da opprettes en forbindelse fra AIS-programmet med Kafka-serveren som har blitt satt opp på samme maskin. Kafka muliggjør at vi kan «lytte» på AIS-meldinger som blir sendt ut på IP-adressen til serveren til kartverket. Alle AIS-meldinger sendes ut, og lagres midlertidig i filen ais_data.txt, for så å bli dekodet som illustrert i tabell 3-3. Etter dekodingen

sjekkes meldingstypen, og hvis det er av interesse, manipuleres og filtreres AIS-dataen, for så å bli lagret i processed_ais_data.txt.

Hvis AIS-dataen er korrupt, (som skjer hvis fartøyet har et svakt signal) eller hvis meldingstypen ikke er av interesse, lagres den i filen failed_decoding.txt. Dette har som hensikt å tilrettelegge for feilsøking.

3.4 Kommunikasjon mellom RPI og Computer

Som kjent benyttes det to hovedkomponenter i systemet: For det første har vi en Raspberry Pi som er utstyrt med et kamera. Denne enheten er ansvarlig for å utføre objektdeteksjon og sending av lagret deteksjonsdata. For det andre anvender vi en bærbar datamaskin som håndterer innsamling av relevant AIS-data samt behandling, sammenligning og lagring av disse dataene til en database.

For å muliggjøre innhenting av deteksjonsdataen fra RPI'en til den bærbare computeren kreves det en form for direkte kommunikasjonsmetode. Vi kom tidlig frem til at Flask-server var veien å gå, grunnet dens kompatibilitet med Python, dens gode API (Application Programming Interfaces) (Flask, 2023) og dens skalerbarhet.

Flask-serveren kjøres på den bærbare computeren. Når begge enhetene er tilkoblet samme nettverk og RPI'en har koblet seg opp mot computerens IP-adresse, kan RPI'en enkelt overføre deteksjonsmappene så fort mappen er ferdig zippet. Dette skjer automatisk, i den hensikt å oppnå et mest mulig effektivt system.

3.5 Sammenlikning og sammenfatning av data

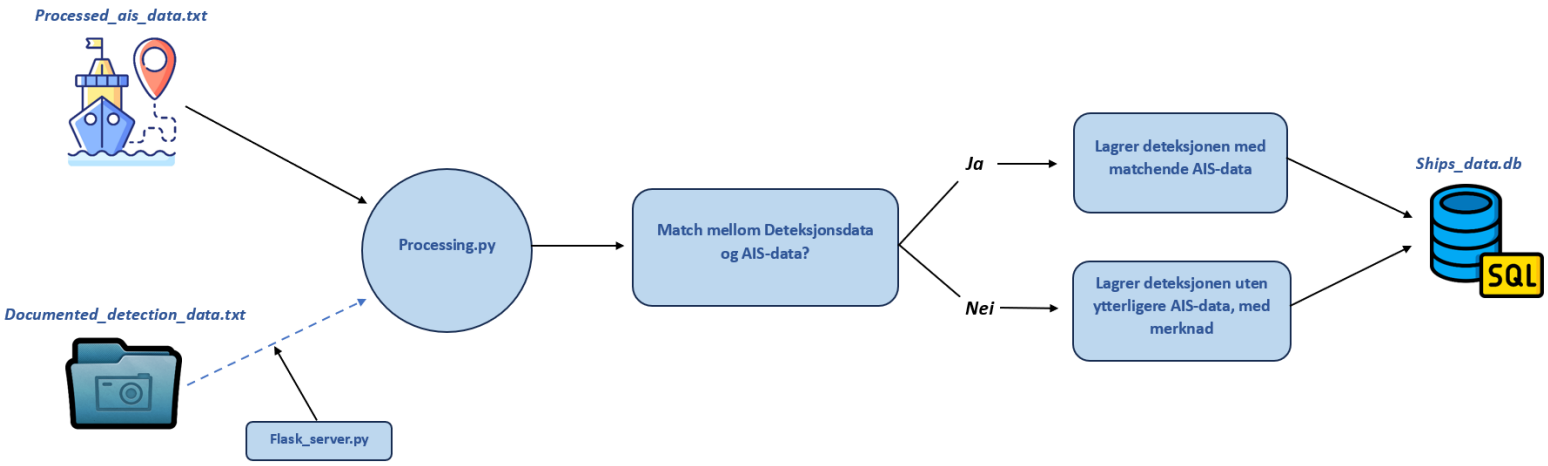
Som nevnt tidligere innhenter den bærbare computeren AIS-data som behandles og lagres på maskinen. Samtidig mottar computeren zippede mapper med deteksjoner over Flask-serveren, fra RPI'en. Når deteksjonsdataen mottas fra RPI'en, sammenliknes dataen med lagret AIS-data.

Som nevnt tidligere besitter hver deteksjonsmappe en tekstfil i tillegg til bilder av deteksjonen. Deteksjonsdataen inneholder blant annet data om den estimerte posisjonen til fartøyet, samt tidspunktet bildet ble tatt. Filen består av data fra hvert enkelt bilde i mappen. Når dataen sammenliknes med AIS-dataen, er dette basert på posisjon og tidspunkt. Koden (se vedlegg D) sammenlikner simultant posisjons- og tidsdata fra tekstfilen med lagrede AIS-meldinger.

Oppdateringsfrekvensen til AIS-utsendelser fra fartøy er, som illustrer i tabell 2-4, er ikke på et sekund, og dermed er det nødvendig å legge inn en buffer for å oppnå match. Bufferen er 30 sekunder på tidspunktet, og 0.0005° på både lengde- og breddegrad. På posisjonen tilsvarer dette en radius på 55 meter.

Hvis det blir match mellom deteksjons- og AIS-data, lagres deteksjonsmappen med tilhørende AIS-data i SQL-databasen. Hvis fartøyet ikke sender ut AIS-data, eller deteksjonsdataen ikke samsvarer med AIS-dataen, lagres likevel deteksjonen i databasen, med en merknad om at fartøyet mangler AIS-data. Fartøy uten AIS-data er også av interesse for datasettet, selv om innhenting av informasjon om fartøyet blir en manuell prosess.

På figur 3-17 illustreres dataflyten til systemet.



Figur 3-13: Dataflyt for Processing.py

4 Tester og resultater

Oppgaven har nå tatt for seg teorien som legger grunnlaget, og selve implementeringen av systemet for dette prosjektet. Videre i denne delen skal oppgaven ta for seg testene som har blitt gjort av systemet. Testene har blitt gjort på forskjellig tidspunkt i prosjektet, og hensikten med denne delen av oppgaven er å få frem utviklingen av systemet og resultatet. For å oppnå målene i tabell 1-1 må systemet klare å detektere og lagre informasjon, samle relevante AIS-meldinger, identifisere korresponderende data og lagre informasjonen i en database. Testene er derfor delt inn i disse underkapitlene, objektdeteksjon og lagring, AIS innhenting, sammenfatning og database.

4.1 Objektdeteksjon og lagring

Denne delen av testingen skal ta for seg tester av maskinlæringsmodellene og programmene for objektdeteksjon på RPI'en.

4.1.1 mAP for den trente modellen

Under treningen av ssd_mobilenet_v2 FPN-lite ble det utført en mean average precision (mAP) test på det tilhørende testsettet. Dette ble gjort som en del av google colab notatblokken (Juras, TensorFlow Lite Object Detection API in Colab, 2023). Resultatet ble 73,25% noe som er vesentlig mye høyere enn alle modellene hadde på COCO 2017 testsettet. Se figur 4-1 for å se tall fra testen.

```
Calculating mAP at 0.50 IoU threshold...
96.77% = boat AP
mAP = 96.77%
Calculating mAP at 0.55 IoU threshold...
96.27% = boat AP
mAP = 96.27%
Calculating mAP at 0.60 IoU threshold...
96.27% = boat AP
mAP = 96.27%
Calculating mAP at 0.65 IoU threshold...
96.27% = boat AP
mAP = 96.27%
Calculating mAP at 0.70 IoU threshold...
92.73% = boat AP
mAP = 92.73%
Calculating mAP at 0.75 IoU threshold...
85.15% = boat AP
mAP = 85.15%
Calculating mAP at 0.80 IoU threshold...
78.53% = boat AP
mAP = 78.53%
Calculating mAP at 0.85 IoU threshold...
53.73% = boat AP
mAP = 53.73%
Calculating mAP at 0.90 IoU threshold...
36.46% = boat AP
mAP = 36.46%
Calculating mAP at 0.95 IoU threshold...
0.29% = boat AP
mAP = 0.29%

***mAP Results***

Class           Average mAP @ 0.5:0.95
----------------------------------------
boat            73.25%

Overall         73.25%
```

Figur 4-1: Utklipp fra mAP test på den trente modellen. Overall resultat er 73.25%, IoU threshold er sannsynlighet for at det er rikitg objekt

4.1.2 FPS og sannsynlighet

Programmene regner selv ut FPS (frames per second) under kjøring av programmene. Dette sier oss altså hvor mange bilder som blir tatt og prosessert hvert sekund i kamera-strømmen. For dette prosjektet kan det diskuteres hvor viktig denne faktoren er, da far-tøyene bruker relativt lang tid forbi kameraet. Hvis systemet selv hadde vært i bevegelse, som for eksempel om bord på et skip, er man mer avhengig av å fange opp alle bevegelser hurtigere da oversikten i kamerabildet kan være varierende. Men uansett sier FPS'en oss en del om hvor krevende modellene er å kjøre, noe som er av interesse.

Det har også blitt regnet ut en gjennomsnittlig sannsynlighet for alle modellene, altså hvor sikre modellene er på objektet de detekterer. Dette er regnet ut som et gjennomsnitt av deteksjonene som er lagret under en test av hver enkelt modell. Alle testene er gjort med tilnærmet like lysforhold, på samme båten (hurtigbåten som kjører forbi jevnlig), for at testene skal reflektere et så reelt resultat som mulig. Dette er av interesse for å undersøke

«terskelverdien» for lagring av deteksjon som er nevnt i kapittel 3.2.4. Fullstendig utregning og presentasjon av tallene er vist i vedlegg H. Se figur 4-2 for to eksempelbilder fra testen, og tabell 4-1 for resultatene.

Figur 4-2: Bilder tatt under tester. Det øverste bildet er ssd_mobilenet_v2 FPN-lite (videretrent modell), det nederste bildet er efficientDet_lite 1, men ser likt ut for 0 og 2. Fargen på boksene har ingen betydning, er bare forskjellig for modellene.

	efficient-Det_lite 0	efficient-Det_lite 1	efficient-Det_lite 2	ssd_mobilenet_v2 FPN-lite
FPS	5,3	3,5	2,4	2,0
SANNSYNLIGHET	44.38%	53.75%	58,63%	81,13%

Tabell 4-1: Resultater fra FPS og sannsynlighetsutregning

Resultatene forteller oss at de mer krevende programmene får en tregere datastrøm. Videre er det også et relativt stort sprik i sannsynlighetene for deteksjon hos de forskjellige modellene. Det skal adresseres at for større fartøy har alle modellene større sannsynlighet,

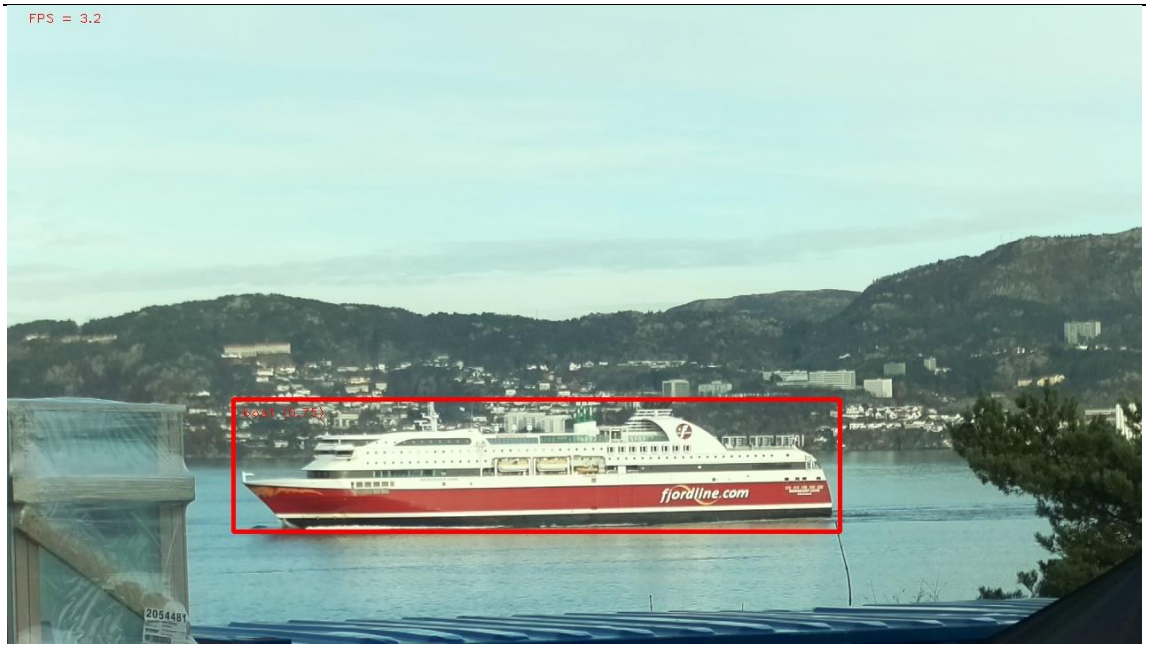
da det er lettere å detektere båter som dekker mer av kamerabildet. Disse resultatene understøtter også at terskelverdien på 50%, ikke er en urimelig grense å sette. I vedlegg H ser man også at alle modellene har mange deteksjoner godt over 50%, selv om den sammenlagte summen er lavere. Allikevel klarer alle modellene å detektere og ramme inn fartøyene med liten feilmargin i store deler av tilfellene. Det ble derfor videre testet nøyaktigheten de forskjellige modellene klarte å ramme inn fartøyene med, da dette er viktig når man lager datasett.

4.1.3 Tester av modellene

For å finne ut mer om hvordan modellene presterer under kjøring, ble det gjennomført testing i likhet med testen i forrige kapittel. I disse testene var hensikten å se hvor godt modellene klarte å markere eller ramme inn båtene som passerer, altså hvor godt de klarer å følge båtene mens de passerer kameraet.

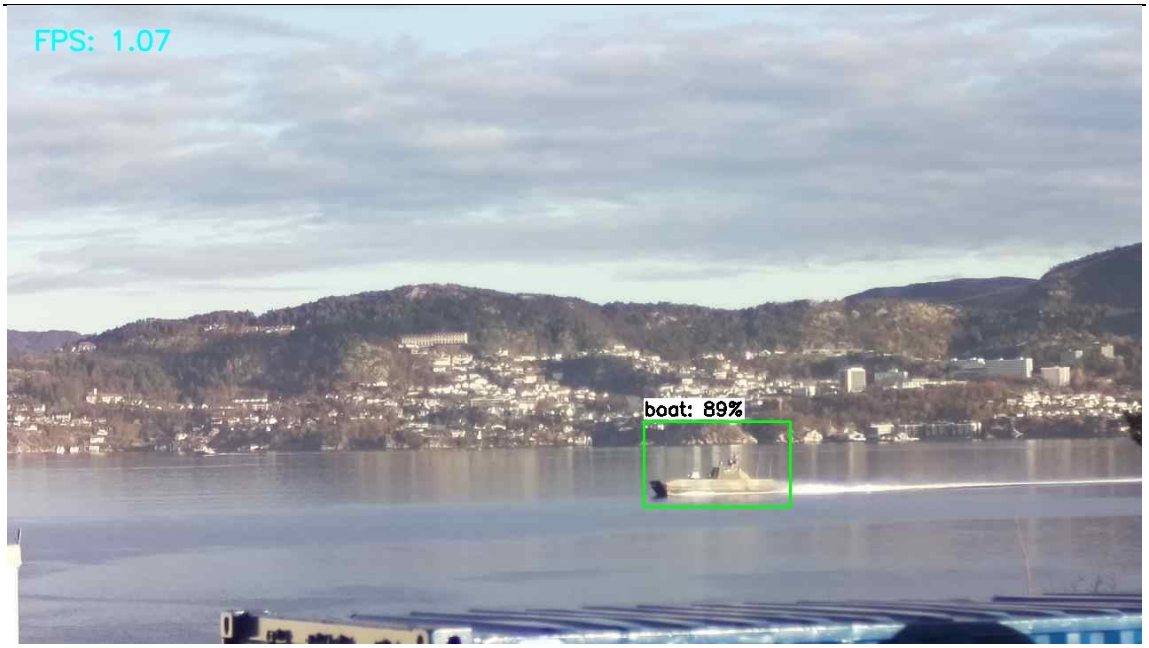
De to viktigste faktorene som påvirket testene var størrelse og fart. Når båtene var av større størrelse var det lettere for modellene og nøyaktig detektere dem, samtidig som farten til fartøyet spilte en mindre rolle. Alle modellene klarte å ramme inn disse fartøyene noenlunde likt, det kan som nevnt tidligere naturlig nok skyldes at et større område av kamerabildet dekkes av båten. Videre ble derfor modellens prestasjoner mål ut ifra hvor godt de klarer å kjenne igjen mindre båter.

Figur 4-3: ssd_mobilenet_v2 FPN-lite (videretrent modell) på et større fartøy. Fargen på annoteringen er ikke av betydning

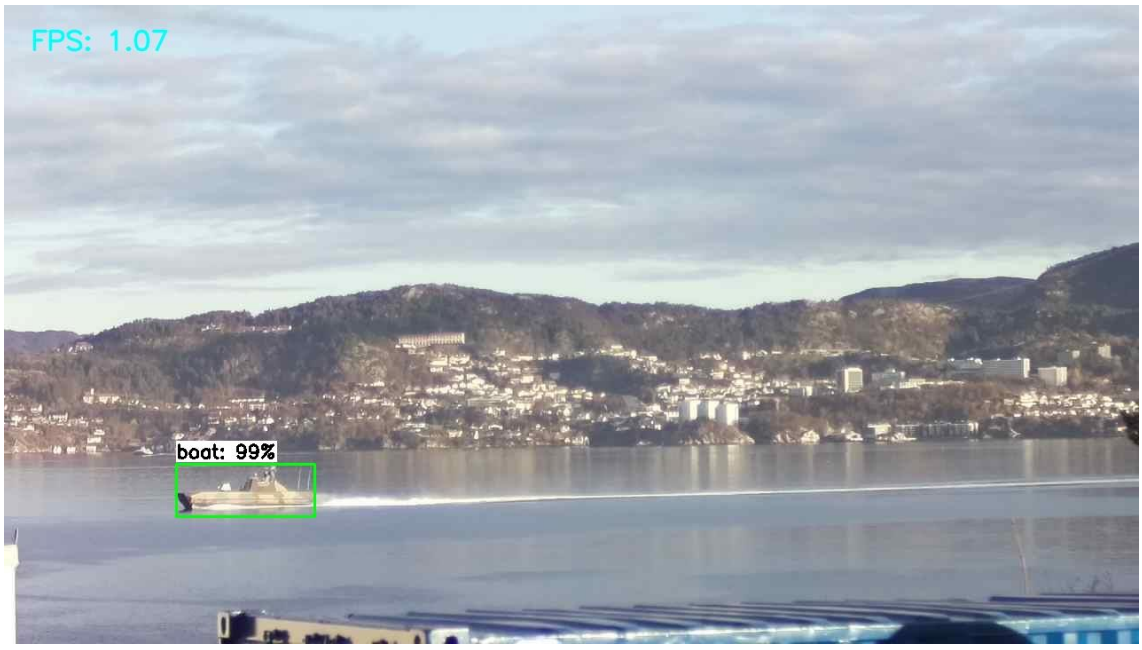


Figur 4-4: efficientdet_lite1 på et større fartøy. Fargen på annoteringen er ikke av betydning

Når modellene skulle testes på mindre båter, ble det utført tester på båter i størrelse med hurtigbåten, vist i figur 4-2. Her kom det fort frem at modellene som har lavere FPS slet med å få like mange deteksjoner på fartøyene som hadde høy fart. Det er plausibelt å anta at dette skyldes at oppløsningen blir dårligere, når modellen ikke får inn like mange bilder i sekundet og båtene beveger seg fort. Dette gjelder spesielt den trente modellen og efficinetdet_lite2 da disse har lavest FPS, men det skal sies at når det først ble gjort deteksjoner, var disse nøyaktige. Den trente modellen har alltid høyere sannsynlighet, men slet mer enn efficientdet_lite2 med å få alle boksene godt tilpasset.



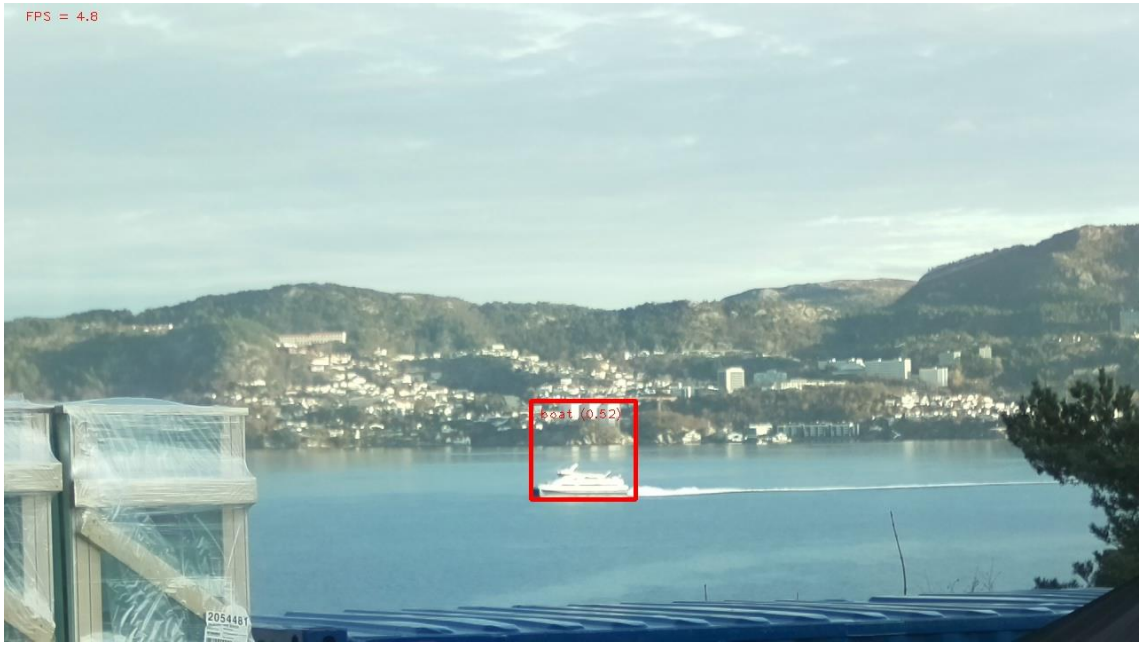
Figur 4-5: Deteksjon av en korvett i 40+ knop, høy sannsynlighet, men unøyaktig annotering



Figur 4-6: Deteksjon av samme korvett som figur 4-5, men som en tilnærmet perfekt deteksjon

Bildene i figur 4-5 og figur 4-6 viser at modellen som er trent, og i likhet med efficientdet_lite2 har høye sannsynligheter, uansett om deteksjonene ikke skjer like hyppig og sporadisk kan være unøyaktig. Det samme problemet ble også detektert ved bruk av efficientdet_lite0, selv om denne modellen har høyest FPS har den også dårligere mAP

og sannsynlighet (tabell 4-1). Denne modellen er dårligst trent og dette medførte også at den slet med å ramme inn båtene når størrelsen var liten og farten ble høy.



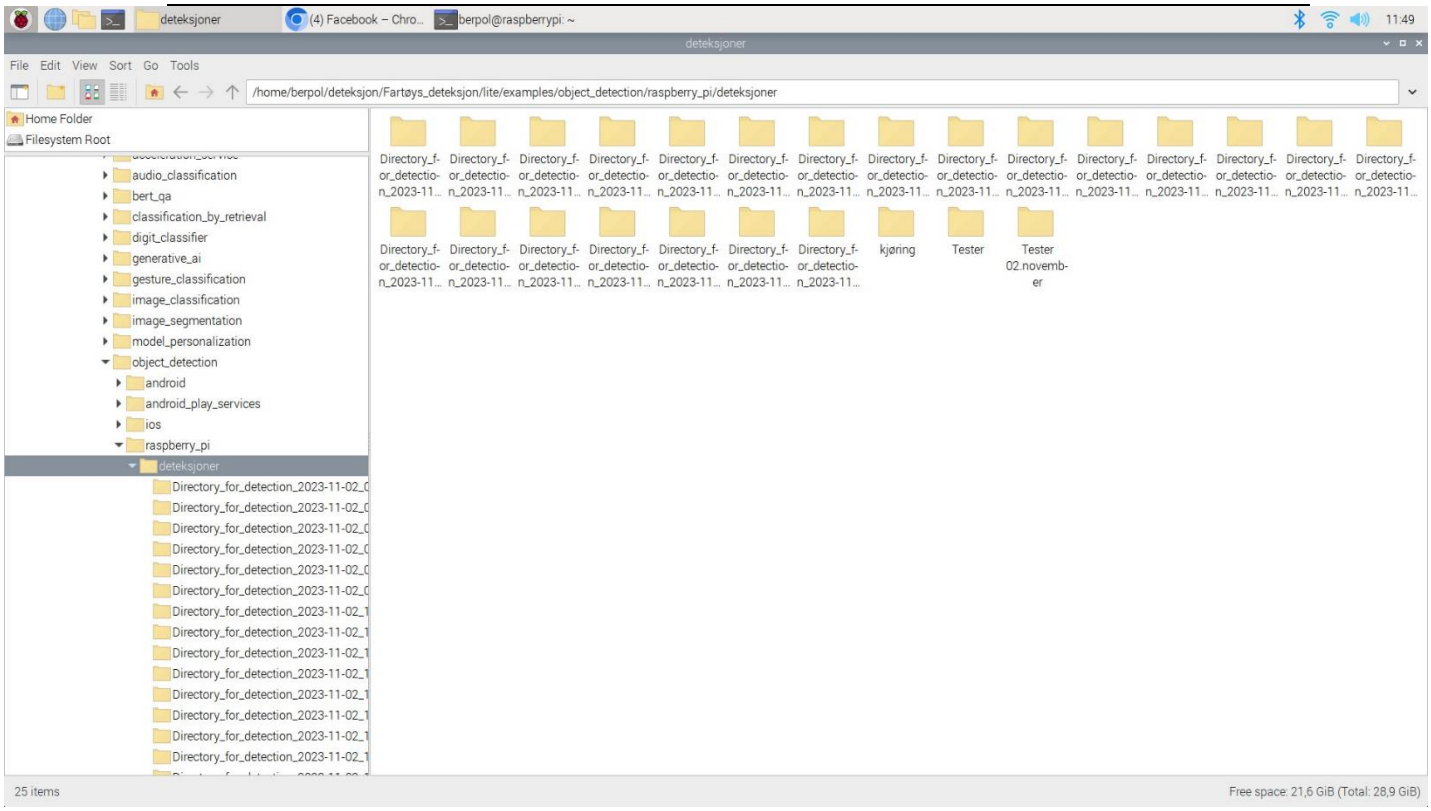
Figur 4-7: Eksempel deteksjon gjort med efficientdet_lite0, dårlig annotering

I konklusjon reflekterer resultatene at alle modellene er i stand til å detektere så å si alle båter av rimelig størrelse, altså ikke veldig små fritidsbåter, og med forskjellig hastighet. Dette resulterer i at man får deteksjonsdata uansett, men at dataen kan variere i mengde og nøyaktighet. Dette er faktorer som er viktige å ta hensyn til, da prosjektets overordnete mål er å hente inn så gode datasett som mulig. For å oppnå nøyaktige datasett er mange bilder og presise annoteringsbokser avgjørende.

4.1.4 Lagring under programkjøring på RPI

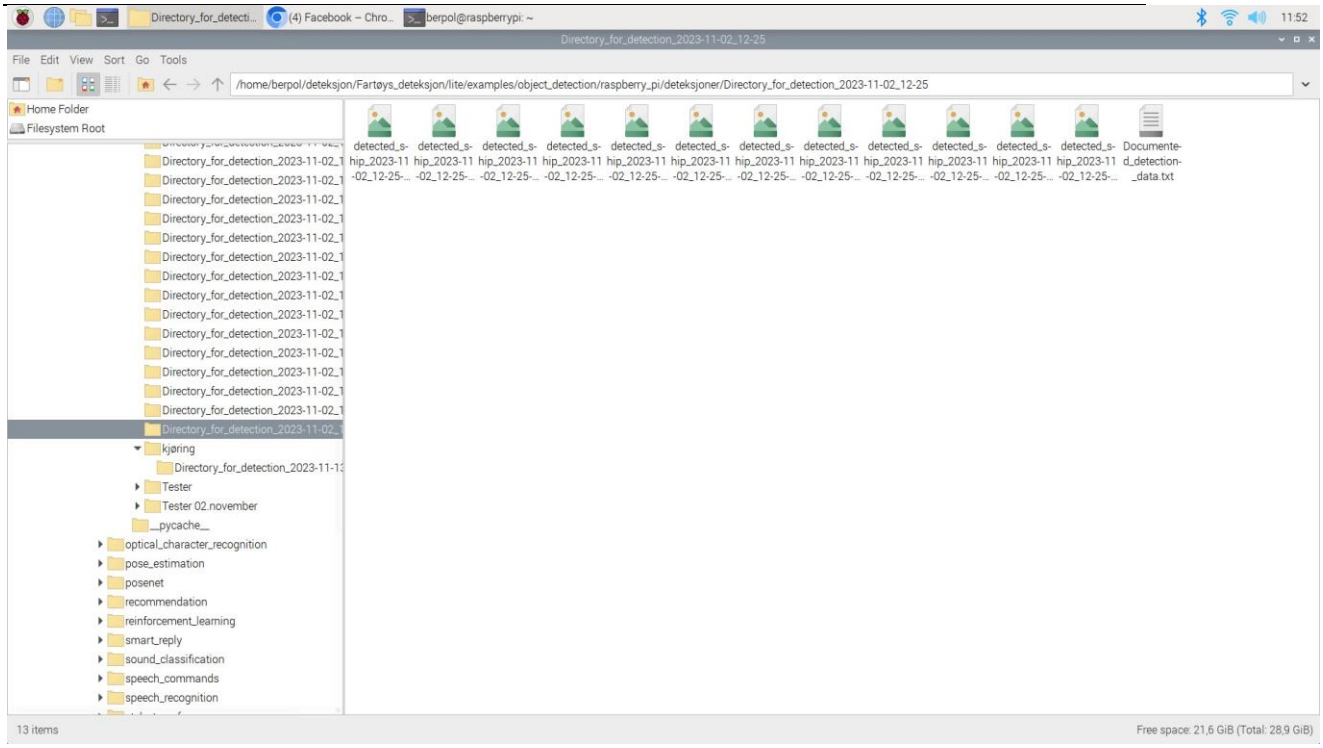
Det er viktig at deteksjonene som lagres blir lagret på en ryddig og oversiktlig måte. I implementeringen ble det beskrevet hvordan informasjonen hentes inn, og hva som lagres i deteksjonsmappen. At informasjonen blir lagret likt for alle deteksjoner er kritisk, da det er essensielt for å kunne hente ut dataen og bruke den under AIS sammenlikningen.

Når det blir gjort en deteksjon av en båt blir det opprettet en mappe inne i «deteksjoner»-mappen. Disse mappene har ulike navn, og navnene er «directory_for_detection_{timestamp}», her får alle mappene et unikt tidsstempel som gjør dem ulike.



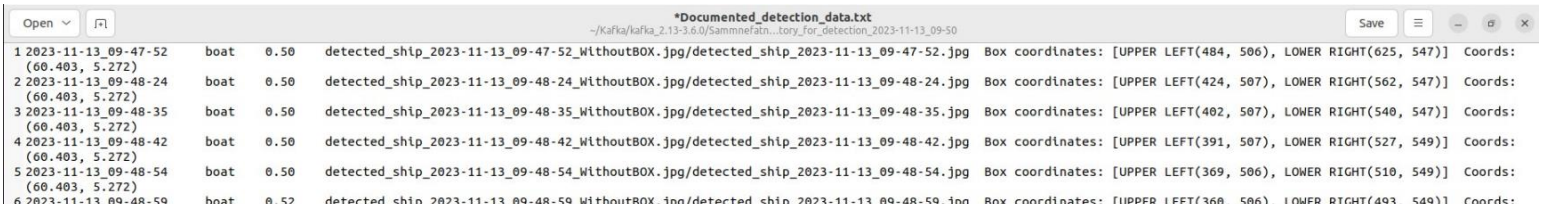
Figur 4-8: Eksempel fra «deteksjoner»-mappen. Her ser man 22 deteksjoner.

Inne i mappene lagres det bilder av deteksjonen hvert femte sekund båten er i kamerabildet. Disse bildene lagres både med og uten annoteringsrammen. Bildet med rammen er fint for å kunne se deteksjonen, men for å kunne bruke bildene som et datasett til trening er man avhengig å ha bildene uten rammen rundt objektet. Sammen med bildene i mappen lagres det en tekstfil som inneholder deteksjonsinformasjonen.



Figur 4-9: Eksempel mappe, med lagring av 6 deteksjoner, derav 12 bilder, og en tekstfil

Tekstfilen er strukturert på en bestemt måte og inneholder informasjonen som er beskrevet i implementeringen. Navnene på bildene kommer opp sammen med informasjonen, slik at det går an å navigere seg frem til bildet informasjonen hører til. I figur 4-10 ser man et eksempel på den lagrede informasjonen, og sammen med bildene er dette resultatet som levers videre over til pc'en ved hjelp av Flask-serveren.

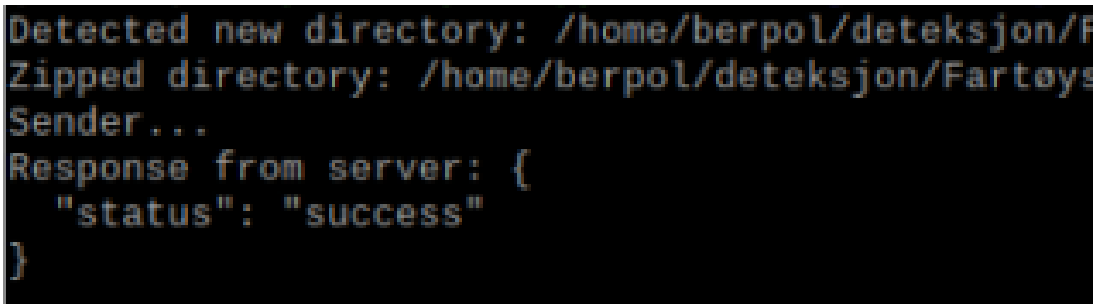


Figur 4-10: Eksempel på deteksjonsdatafilen, her åpnet på Pc'en etter sending fra RPI

4.1.5 Sending av deteksjonsdata

Når en deteksjonsmappe blir opprettet vil programmene data_overføring.py og data_sender.py (uavhengig av hvilken modell som blir kjørt) oppdage at det har blitt opprettet en

deteksjon i «deteksjoner»-mappen tilhørende programfilene. Etter programmene har oppdaget dette går det litt tid før mappen med deteksjonsdata blir sendt, og prosessen blir skrevet ut i terminalvinduet hvis man har en skjerm koblet til RPI'en.



Figur 4-11: Terminalvindu under sending på RPI

Som man kan se på utklippet fra testen av sending på figur 4-11, kommer det en respons fra serveren. Denne responsen blir sendt tilbake fra pc'en og forteller oss om den har mottatt deteksjons mappen med «success», eller at det har oppstått en feil med forskjellige feilmeldinger avhengig av hvor feilen har skjedd i prosessen. Programmet prøver ikke på nytt før det kommer en ny datapakke, det vil si at en datapakke ikke blir sendt ved «error». Under testen ble det erfart enhetene må være koblet til et nettverk som tillater de å kommunisere sammen, og IP-adressene må oppdateres ved skifte av nett. Løsningen for å oppnå en stabil oppkobling for kommunikasjon var å koble begge enhetene opp mot en mobil 2,4G ruter som var tilgjengelig på skolen.

4.2 Innhenting av AIS-data

Som en klient/bruker av programmet, kan det startes en overvåkning ved å definere et område av interesse, som illustrert på figur 3-11, og deretter kjøre programmet som kontinuerlig innhenter data. Informasjon om behandlet data blir illustrert i terminalvinduet, som illustrert på figur 4-8.

```
Totalt antall meldinger lest: 187
Relevante meldinger: 1
Meldinger som feilet: 91
Møtte meldingstyper: [24, 1, 18, 3]
Totalt antall meldinger lest: 269
Relevante meldinger: 0
Meldinger som feilet: 104
Møtte meldingstyper: [1, 3, 8, 18, 21, 24]
Totalt antall meldinger lest: 201
Relevante meldinger: 0
Meldinger som feilet: 73
Møtte meldingstyper: [1, 3, 18, 21, 24]
Totalt antall meldinger lest: 265
Relevante meldinger: 1
Meldinger som feilet: 94
```

Figur 4-12: Visualisering av AIS-filetering

Visualiseringen gir en oversikt over totalt antall AIS-meldinger som kommer inn, hvor mange som er innenfor område av interesse, hvor mange meldinger som er irrelevante eller korrupte, og hvilke meldingstyper som kommer inn.

Gjennom testing erfarte vi at visualiseringen også er et nyttig verktøy for feilsøking, da den hvert femte sekund bekrefter om AIS-data innhentes. Dette effektiviserte feilsøkingen for oss, og gjorde systemet mer oversiktlig. Man kan eksempelvis se når internett-utfordringer oppstår, da dette påvirker totalt antall meldinger som leses.

Relevante meldinger filtreres og lagres som nevnt i processed_ais_data.txt, som illustrert på figur 4-12.

Open

*processed_ais_data.txt
~/Kafka/kafka_2.13-3.6.0/Innhenting_Konvertering

```
 1 {'id': 1, 'mmsi': 258416000, 'y': 60.403, 'x': 5.282, 'timestamp': '2023-11-13_08-24-47'}
 2 {'id': 1, 'mmsi': 258416000, 'y': 60.404, 'x': 5.265, 'timestamp': '2023-11-13_08-55-16'}
 3 {'id': 1, 'mmsi': 258108500, 'y': 60.402, 'x': 5.271, 'timestamp': '2023-11-13_08-55-27'}
 4 {'id': 3, 'mmsi': 258416000, 'y': 60.403, 'x': 5.297, 'timestamp': '2023-11-13_09-21-42'}
 5 {'id': 1, 'mmsi': 258416000, 'y': 60.404, 'x': 5.272, 'timestamp': '2023-11-13_09-23-44'}
 6 {'id': 1, 'mmsi': 257736000, 'y': 60.404, 'x': 5.309, 'timestamp': '2023-11-13_09-28-21'}
 7 {'id': 3, 'mmsi': 257736000, 'y': 60.403, 'x': 5.306, 'timestamp': '2023-11-13_09-30-23'}
 8 {'id': 1, 'mmsi': 258416000, 'y': 60.403, 'x': 5.283, 'timestamp': '2023-11-13_09-35-48'}
 9 {'id': 1, 'mmsi': 258738000, 'y': 60.405, 'x': 5.307, 'timestamp': '2023-11-13_09-38-02'}
10 {'id': 1, 'mmsi': 258738000, 'y': 60.407, 'x': 5.303, 'timestamp': '2023-11-13_09-40-10'}
11 {'id': 1, 'mmsi': 257706000, 'y': 60.401, 'x': 5.274, 'timestamp': '2023-11-13_09-45-31'}
12 {'id': 1, 'mmsi': 257706000, 'y': 60.402, 'x': 5.265, 'timestamp': '2023-11-13_09-47-28'}
13 {'id': 3, 'mmsi': 258242000, 'y': 60.403, 'x': 5.292, 'timestamp': '2023-11-13_09-52-35'}
14 {'id': 1, 'mmsi': 304619000, 'y': 60.398, 'x': 5.273, 'timestamp': '2023-11-13_10-00-27'}
```

Figur 4-13: Eksempeldata fra processed_ais_data.txt

Første gang vi benyttet oss av dette systemet for datainnhenting, lagret systemet kun første deteksjon av hvert fartøy. Noe som førte til at det sjeldnere ble match mellom deteksjon og AIS-data. Løsningen var å samle inn all AIS-data som fartøyet sendte ut innenfor interesseområdet. Dette kommer tydelig frem i figur 4-12, hvor samme MMSI-nummer befinner seg flere steder i dokumentet.

Gjennom testing erfarte vi videre utfordringer ved fartøyers AIS-innmelding. Grunnet bruk av kystverkets åpne server, var det lav frekvens på fartøyers AIS-innmelding. Dette kommer tydelig frem på figur 4-12, hvor melding nummer fire og fem, som er samme fartøy, har et tidsmellomrom på to minutter. Dette erfarte vi som en utfordring, da det gjennomsnittlig tok en båt to minutter å passere kameraets bildesektor. Dermed forsøkte vi å anskaffe tilgang på kystverkets private AIS-server, noe som ble mer utfordrende enn anslått da vi ikke hadde tilgang på en fast IP-adresse i dette prosjektet.

Resultatet ble dermed at vi fortsatte å benytte oss av kystverkets åpne server. Som beskrevet i avsnitt 3.2.4, valgte vi å benytte oss av en statisk GPS-posisjon midt i kamerabildet. Lav AIS-innmelding var en årsakene for dette valget.

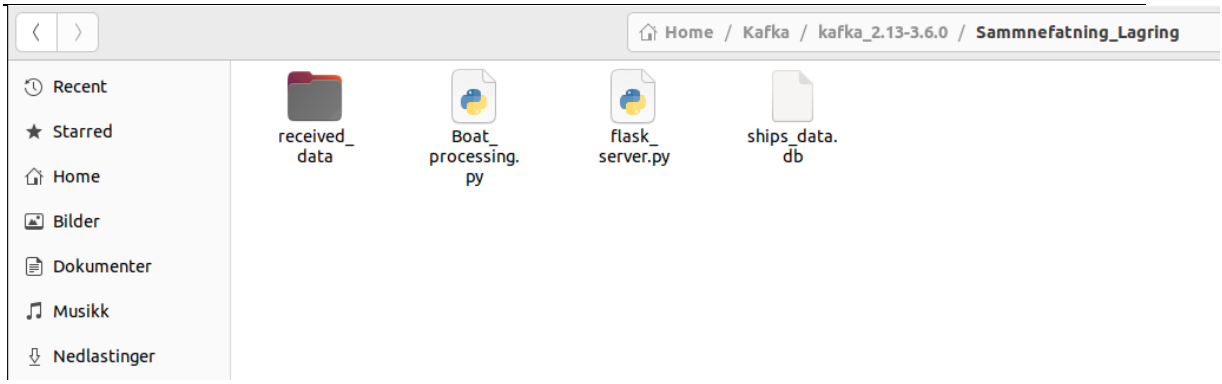
4.3 Sammenfatning og database

De tidligere delkapitlene har tatt for seg testing og resultater fra datainnhentingen. I dette delkapittelet skal det bli presentert videre tester og resultater av sammenfatting og lagring.

4.3.1 Hovedprogrammet

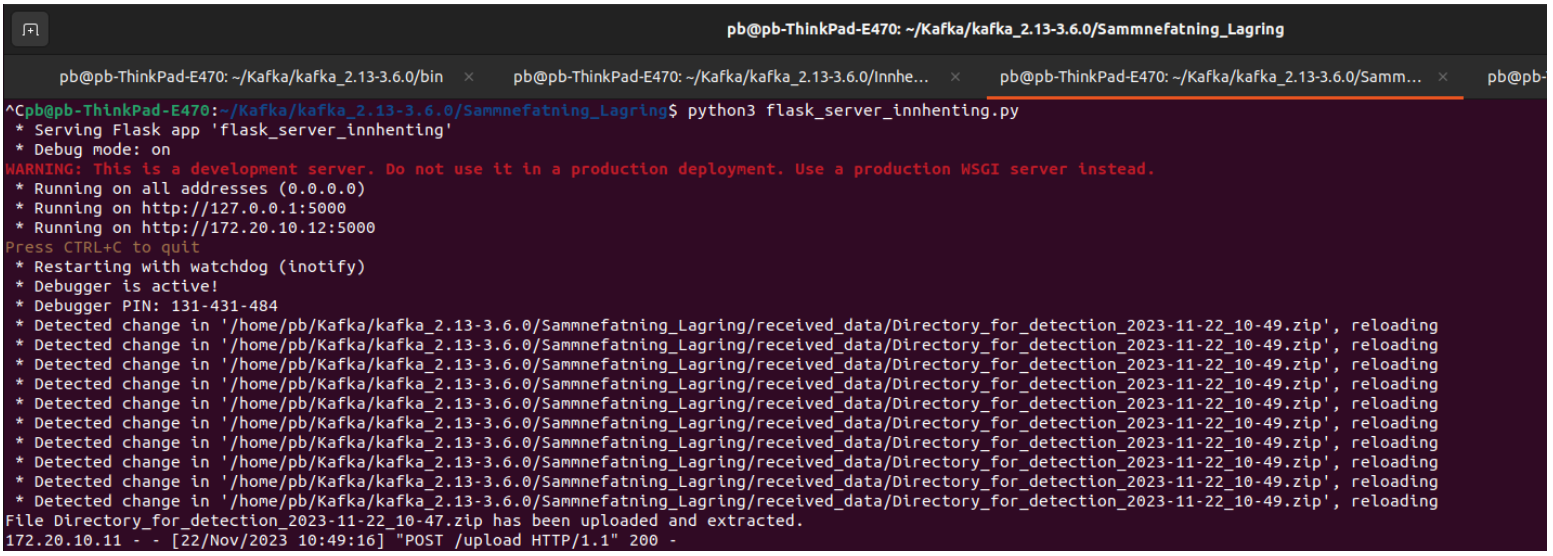
Hovedprogrammet har i oppgave å sammenlikne innhentet data og lagre det i databasen. Flask-serveren muliggjør datainnhenting fra RPI'en, slik at alt av data lagres lokalt på den bærbare computeren. Ordinært var alt av prosesser samlet i kun et program, som gjorde alt fra oppsett av Flask-server og SQL-database, til sammenlikning og lagring. Det ble erfart under testing at dette var uoversiktlig, og noe som medførte utfordringer ved feilsøking da det oppstod feil.

Det ble følgelig valgt å dele programmet opp i to, hvor koden ved navn flask_server.py har som funksjon å drifte Flask-serveren og lagre overført objektdeteksjons data. Processing.py drifter databasen, sammenlikner og lagrer data i databasen. Systemets filstruktur er illustrert på figur 4-13.



Figur 4-14: Filstruktur for datasammenfatning

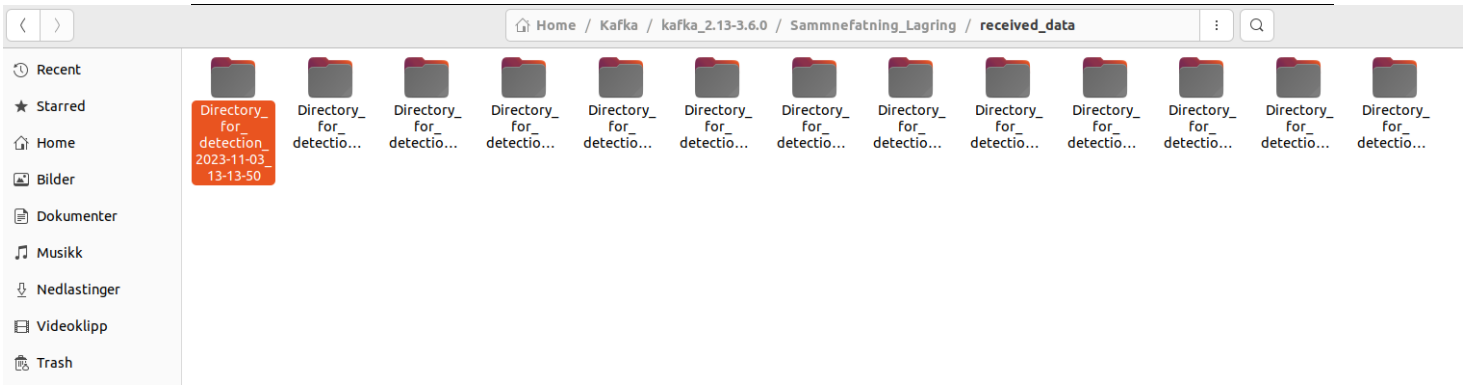
Når flask_server.py programmet kjøres, er den bærbare computeren mottagelig for data fra RPI'en. Når en deteksjonsmappe sendes fra RPI'en, kommer det en notifikasjon i terminalvinduet som illustrert på figur 4-15. Her informeres det om at filen har blitt lastet opp på serveren av RPI'en og overført til hosten. Terminalvinduet indikerer også hvilke IP-adresser Flask-serveren kjører på, noe som belyser hvilke IP-adresse RPI'en må benytte for å overføre ønsket data.



```
^Cpb@pb-ThinkPad-E470:~/Kafka/kafka_2.13-3.6.0/Sammnefatning_Lagring$ python3 flask_server_innhenting.py
 * Serving Flask app 'flask_server_innhenting'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://172.20.10.12:5000
Press CTRL+C to quit
 * Restarting with watchdog (inotify)
 * Debugger is active!
 * Debugger PIN: 131-431-484
 * Detected change in '/home/pb/Kafka/kafka_2.13-3.6.0/Sammnefatning_Lagring/received_data/Directory_for_detection_2023-11-22_10-49.zip', reloading
 * Detected change in '/home/pb/Kafka/kafka_2.13-3.6.0/Sammnefatning_Lagring/received_data/Directory_for_detection_2023-11-22_10-49.zip', reloading
 * Detected change in '/home/pb/Kafka/kafka_2.13-3.6.0/Sammnefatning_Lagring/received_data/Directory_for_detection_2023-11-22_10-49.zip', reloading
 * Detected change in '/home/pb/Kafka/kafka_2.13-3.6.0/Sammnefatning_Lagring/received_data/Directory_for_detection_2023-11-22_10-49.zip', reloading
 * Detected change in '/home/pb/Kafka/kafka_2.13-3.6.0/Sammnefatning_Lagring/received_data/Directory_for_detection_2023-11-22_10-49.zip', reloading
 * Detected change in '/home/pb/Kafka/kafka_2.13-3.6.0/Sammnefatning_Lagring/received_data/Directory_for_detection_2023-11-22_10-49.zip', reloading
 * Detected change in '/home/pb/Kafka/kafka_2.13-3.6.0/Sammnefatning_Lagring/received_data/Directory_for_detection_2023-11-22_10-49.zip', reloading
 * Detected change in '/home/pb/Kafka/kafka_2.13-3.6.0/Sammnefatning_Lagring/received_data/Directory_for_detection_2023-11-22_10-49.zip', reloading
 * Detected change in '/home/pb/Kafka/kafka_2.13-3.6.0/Sammnefatning_Lagring/received_data/Directory_for_detection_2023-11-22_10-49.zip', reloading
 * Detected change in '/home/pb/Kafka/kafka_2.13-3.6.0/Sammnefatning_Lagring/received_data/Directory_for_detection_2023-11-22_10-49.zip', reloading
File Directory_for_detection_2023-11-22_10-47.zip has been uploaded and extracted.
172.20.10.11 - - [22/Nov/2023 10:49:16] "POST /upload HTTP/1.1" 200 -
```

Figur 4-15: Terminalvindu på computer under kjøring av Flask-server

Det ble valgt å lagre alle deteksjoner i mappen received_data. Her ble i første omgang både den zippede og unzippede mappen lagret, men etter tester ble det konkludert med at det ikke var behov for å lagre den zippede mappen. Det ble dermed integrert en funksjon i koden som slettet den zippede mappen så fort den hadde blitt unzippet, for å gjøre det lettere å holde oversikt. Received_data ble dermed seende ut som illustert på figur 4-16.



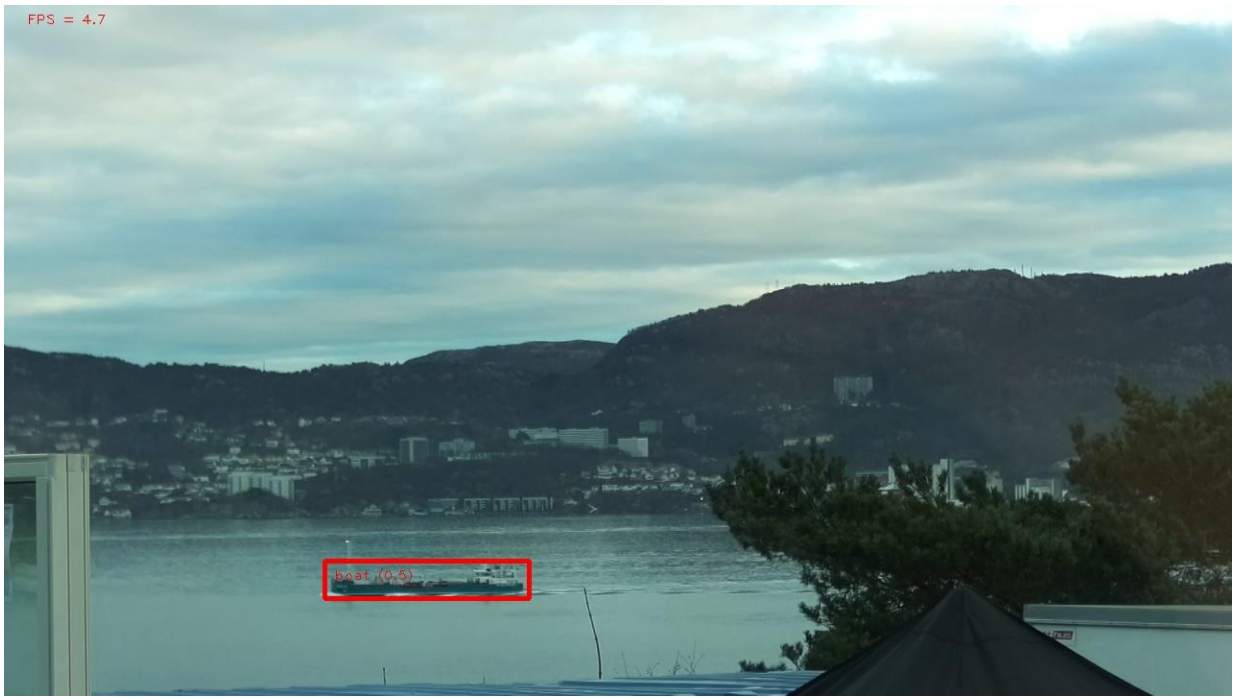
Figur 4-16: Lagring av deteksjonsmapper på computer

Når det kom til sammenligning av deteksjonsdata og AIS-data det erfart noen utfordringer under testing. Som nevnt i delkapittel 3.3.3, implementerete vi en buffer på både tidspunkt og posisjon for å muliggjøre match mellom deteksjons- og AIS-data. Årssaken er at innmedlingsfrekvensen til AIS-dataen er sjeldnere enn hvert sekund, som illustrert i tabell 2-4. Dermed er det urealistisk å oppnå deteksjoner på sekundet, og heller ikke 100% nøyaktig posisjon. Løsningen ble implementering av en buffer, som initielt ble satt til 30 sekunder og 0.0005° i både latitude og longitude. Etter videre testing ble det erfart at oppdateringsfrekvensen på AIS-meldingene var lav, det ble derfor vanskelig å matche dataen og dermed måtte vi øke bufferen. Bufferen ble endret til to minutter og 0.002°

Resultatet av bufferendringen var at programmet klarte å finne AIS-informasjon innenfor bufferområene oftere, og det ble lagret deteksjoner med AIS-data. Ved å justere bufferen ble det på en annen side åpnet opp for at presisjonen i dataen ble dårligere, og at sannsynligheten for mismatch mellom deteksjons- og AIS-data økte. Konsekvensen ble altså at sannsynligheten for at en deteksjon får AIS-data tilhørende et annet fartøy økte. Etter videre og nøye testing ble det konkludert med at dette egentlig ikke var et stort problem, da oppsettet til kameraet i dette prosjektet er satt opp til å fokusere på så få fartøy som mulig om gangen. Med dette refereres det til kapittel 3.2.4, der det blir beskrevet at det er fordelaktig å ha kameraet plassert på en slik måte at man får best mulig bilder av båtene. Med andre ord vil da fartøyet som er nærmest kameraet og i fokus av bildet, også være det fartøyet som er nærmes den statiske GPS-posisjonen som er satt, selv om kameraet fanger opp flere fartøy i bakgrunnen. Resultatet av dette var at stort sett alle deteksjoner hvor det var AIS-data tilgjengelig ble matchet opp.

```
pb@pb-ThinkPad-E470:~/Kafka/kafka_2.13-3.6.0/Sammnefatning_Lagring$ python3 Boat_prosessering.py
Directory from objectdetection received: /home/pb/Kafka/kafka_2.13-3.6.0/Sammnefatning_Lagring/received_
Handling new directory: /home/pb/Kafka/kafka_2.13-3.6.0/Sammnefatning_Lagring/received_data/Directory_for
Files in directory: ['detected_ship_2023-11-22_10-47-10.jpg', 'Documented_detection_data.txt', 'detected_
Documented_detection_data.txt found
Reading file...
Match in detection data!
Data extracted
Ship details from AIS: Match found
save_to_db: Saving data...
save_to_db: Data saved.
```

Figur 4-17: Utklipp av terminalvindu under kjøring av programmet, meldinger om hvor dataen er i prosessen kommer opp



Figur 4-18: Eksempel på deteksjon som ble lagret med AIS-informasjon i database

5	2023-11-13 09:23:44	258416000	0	1

Sammnefatning_Lagring/received_data/Directory_for_detection_2023-11-1

1-13_09-25	0.5	[(539, 548), (644, 579)]	60.404	5.27

Figur 4-19: Utdrag fra databasen med informasjon tilhørende figur 4-17.

Figur 4-17 og Figur 4-18 er et eksempel på en deteksjon med AIS-data match. Figur 4-18 er et utdrag fra databasen og informasjonen som står der og det generelle oppsettet for databasen og blir forklart i neste kapittel.

4.3.2 Databasen

I forrige kapittel ser man at lagringen til databasen fungerer som den skal. Dataen blir lagret på en oversiktlig og organisert måte. Dataen blir lagret henholdsvis med tidsstempel, mmsi-nummer, om dataen er lagret med eller uten AIS data, stien til deteksjonsmappen, sannsynlighet, boks-koordinater og til slutt koordinatene. For å få innsyn i databasen ble det lastet ned et SQL-program på pc'en med et tilhørende brukergrensesnitt (DB Browser for SQLite, 2021). Gjennom testing ble det avdekket at programmet oppdaterte seg selv kontinuerlig, og det var mulig å bruke dette brukergrensesnittet til å overvåke når det ble lagret nye datapakker i databasen.

File Edit View Tools Help

New Database | Open Database | Write Changes | Revert Changes | Open Project | Save Project | Attach Database | Close Database

Database Structure | Browse Data | Edit Pragmas | Execute SQL

SHIPS

Table: SHIPS — Filter in any column

	timestamp	mmsi	from_camera	from_ais	image_path	confidence	box_coords	latitude	longitude
1	2023-11-09_21-44-44	NULL	1	0	/home/pb/Kafka/kafka_2.13-3.6.0/Sammnefatning_Lagring/received_data/Directory_for_detection_2023-11-09_21-45	0.54	[(9, 362), (681, 717)]	60.403	5.272
2	2023-11-09_21-50-00	NULL	1	0	/home/pb/Kafka/kafka_2.13-3.6.0/Sammnefatning_Lagring/received_data/Directory_for_detection_2023-11-09_21-50	0.63	[(169, 4), (1000, 715)]	60.403	5.272
3	2023-11-13_09-04-08	NULL	1	0	/home/pb/Kafka/kafka_2.13-3.6.0/Sammnefatning_Lagring/received_data/Directory_for_detection_2023-11-13_09-06	0.54	[(292, 175), (651, 706)]	60.403	5.272
4	2023-11-13 08:55:16	258416000	0	1	/home/pb/Kafka/kafka_2.13-3.6.0/Sammnefatning_Lagring/received_data/Directory_for_detection_2023-11-13_09-10	0.57	[(586, 499), (1234, 718)]	60.404	5.265
5	2023-11-13 09:23:44	258416000	0	1	/home/pb/Kafka/kafka_2.13-3.6.0/Sammnefatning_Lagring/received_data/Directory_for_detection_2023-11-13_09-25	0.5	[(539, 548), (644, 579)]	60.404	5.27
6	2023-11-13 09:23:44	258416000	0	1	/home/pb/Kafka/kafka_2.13-3.6.0/Sammnefatning_Lagring/received_data/Directory_for_detection_2023-11-13_09-25	0.52	[(380, 555), (483, 584)]	60.404	5.27
7	2023-11-13_09-35-12	NULL	1	0	/home/pb/Kafka/kafka_2.13-3.6.0/Sammnefatning_Lagring/received_data/Directory_for_detection_2023-11-13_09-37	0.5	[(167, 551), (268, 593)]	60.403	5.272
8	2023-11-13_09-35-17	NULL	1	0	/home/pb/Kafka/kafka_2.13-3.6.0/Sammnefatning_Lagring/received_data/Directory_for_detection_2023-11-13_09-37	0.5	[(341, 551), (437, 588)]	60.403	5.272
9	2023-11-13 09:45:31	257706000	0	1	/home/pb/Kafka/kafka_2.13-3.6.0/Sammnefatning_Lagring/received_data/Directory_for_detection_2023-11-13_09-47	0.5	[(334, 579), (547, 618)]	60.401	5.274
10	2023-11-13 09:45:31	257706000	0	1	/home/pb/Kafka/kafka_2.13-3.6.0/Sammnefatning_Lagring/received_data/Directory_for_detection_2023-11-13_09-47	0.5	[(211, 582), (420, 620)]	60.401	5.274
11	2023-11-13 09:47:28	257706000	0	1	/home/pb/Kafka/kafka_2.13-3.6.0/Sammnefatning_Lagring/received_data/Directory_for_detection_2023-11-13_09-50	0.5	[(484, 506), (625, 547)]	60.402	5.265
12	2023-11-13 09:47:28	257706000	0	1	/home/pb/Kafka/kafka_2.13-3.6.0/Sammnefatning_Lagring/received_data/Directory_for_detection_2023-11-13_09-50	0.5	[(424, 507), (562, 547)]	60.402	5.265
13	2023-11-13 09:47:28	257706000	0	1	/home/pb/Kafka/kafka_2.13-3.6.0/Sammnefatning_Lagring/received_data/Directory_for_detection_2023-11-13_09-50	0.5	[(402, 507), (540, 547)]	60.402	5.265
14	2023-11-13 09:47:28	257706000	0	1	/home/pb/Kafka/kafka_2.13-3.6.0/Sammnefatning_Lagring/received_data/Directory_for_detection_2023-11-13_09-50	0.5	[(391, 507), (527, 549)]	60.402	5.265
15	2023-11-13 09:47:28	257706000	0	1	/home/pb/Kafka/kafka_2.13-3.6.0/Sammnefatning_Lagring/received_data/Directory_for_detection_2023-11-13_09-50	0.5	[(369, 506), (510, 549)]	60.402	5.265
16	2023-11-13 09:47:28	257706000	0	1	/home/pb/Kafka/kafka_2.13-3.6.0/Sammnefatning_Lagring/received_data/Directory_for_detection_2023-11-13_09-50	0.52	[(360, 506), (493, 549)]	60.402	5.265
17	2023-11-13 09:47:28	257706000	0	1	/home/pb/Kafka/kafka_2.13-3.6.0/Sammnefatning_Lagring/received_data/Directory_for_detection_2023-11-13_09-50	0.5	[(354, 509), (490, 547)]	60.402	5.265
18	2023-11-13 09:47:28	257706000	0	1	/home/pb/Kafka/kafka_2.13-3.6.0/Sammnefatning_Lagring/received_data/Directory_for_detection_2023-11-13_09-50	0.5	[(345, 507), (483, 547)]	60.402	5.265
19	2023-11-13 09:47:28	257706000	0	1	/home/pb/Kafka/kafka_2.13-3.6.0/Sammnefatning_Lagring/received_data/Directory_for_detection_2023-11-13_09-50	0.5	[(333, 505), (471, 548)]	60.402	5.265
20	2023-11-13 09:47:28	257706000	0	1	/home/pb/Kafka/kafka_2.13-3.6.0/Sammnefatning_Lagring/received_data/Directory_for_detection_2023-11-13_09-50	0.5	[(325, 506), (463, 549)]	60.402	5.265
21	2023-11-13_09-49-31	NULL	1	0	/home/pb/Kafka/kafka_2.13-3.6.0/Sammnefatning_Lagring/received_data/Directory_for_detection_2023-11-13_09-50	0.5	[(312, 506), (448, 548)]	60.403	5.272
22	2023-11-13_09-49-37	NULL	1	0	/home/pb/Kafka/kafka_2.13-3.6.0/Sammnefatning_Lagring/received_data/Directory_for_detection_2023-11-13_09-50	0.5	[(302, 506), (433, 549)]	60.403	5.272
23	2023-11-13_09-49-42	NULL	1	0	/home/pb/Kafka/kafka_2.13-3.6.0/Sammnefatning_Lagring/received_data/Directory_for_detection_2023-11-13_09-50	0.5	[(298, 506), (427, 548)]	60.403	5.272
24	2023-11-13_09-49-48	NULL	1	0	/home/pb/Kafka/kafka_2.13-3.6.0/Sammnefatning_Lagring/received_data/Directory_for_detection_2023-11-13_09-50	0.52	[(293, 508), (422, 547)]	60.403	5.272
25	2023-11-13_09-49-54	NULL	1	0	/home/pb/Kafka/kafka_2.13-3.6.0/Sammnefatning_Lagring/received_data/Directory_for_detection_2023-11-13_09-52	0.5	[(281, 508), (414, 547)]	60.403	5.272
26	2023-11-13_09-49-59	NULL	1	0	/home/pb/Kafka/kafka_2.13-3.6.0/Sammnefatning_Lagring/received_data/Directory_for_detection_2023-11-13_09-52	0.52	[(276, 508), (405, 548)]	60.403	5.272
27	2023-11-13_09-50-06	NULL	1	0	/home/pb/Kafka/kafka_2.13-3.6.0/Sammnefatning_Lagring/received_data/Directory_for_detection_2023-11-13_09-52	0.5	[(266, 506), (395, 547)]	60.403	5.272
28	2023-11-13_09-50-21	NULL	1	0	/home/pb/Kafka/kafka_2.13-3.6.0/Sammnefatning_Lagring/received_data/Directory_for_detection_2023-11-13_09-52	0.5	[(244, 511), (373, 548)]	60.403	5.272
29	2023-11-22 10:28:15	257301500	0	1	/home/pb/Kafka/kafka_2.13-3.6.0/Sammnefatning_Lagring/received_data/Directory_for_detection_2023-11-22_10-30	0.8	[(181, 1), (1283, 721)]	60.403	5.302
30	2023-11-22 10:28:15	257301500	0	1	/home/pb/Kafka/kafka_2.13-3.6.0/Sammnefatning_Lagring/received_data/Directory_for_detection_2023-11-22_10-30	0.9	[(181, 0), (1283, 720)]	60.403	5.302
31	2023-11-22 10:46:37	257706000	0	1	/home/pb/Kafka/kafka_2.13-3.6.0/Sammnefatning_Lagring/received_data/Directory_for_detection_2023-11-22_10-49	0.7	[(731, 93), (1271, 723)]	60.404	5.276

Figur 4-20: Utklipp av databasen åpnet i brukergrensesnittet

5 Drøfting

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

5.1 Resultater drøftet opp mot mål for oppgaven

I denne oppgaven har det blitt utviklet et mobilt system for maritim overvåkning og datainnhenting. Produktet skulle utvikle miljøtilpassede treningssett for trening av kunstig intelligens (KI), ved å anvende objektdeteksjon i kombinasjon med AIS-data. Dette har blitt gjort gjennom modifisering av deteksjons-script som utnytter output fra objektdeteksjonsmodellen, og scripts for innhenting og sammenfatning av deteksjons- og AIS-data. Fokuset gjennom hele prosjektet har vært å implementere AIS-data til objektdeteksjon, i den hensikt å automatisere identifikasjon av store deler av deteksjonene, for så å kunne bruke disse som treningsdatasett.

Mål	**Beskrivelse**
Datainnsamling	Kunne automatisk detektere og lagre deteksjonsdata, samt kontinuerlig innhente AIS-data i et avgrenset område.
Sammenligning/identifikasjon	Dataen fra de forskjellige kildene skal kunne håndteres sammen for identifikasjon, samt identifisere deteksjoner som mangler AIS-data
Database	Datapakker med relevant informasjon om en deteksjon skal lagres i en database
Visualisering/overvåkning	Det skal være mulig for en eventuell operatør å se når nye datapakker kommer inn i databasen, samt overvåke systemets drift

Tabell 5-1: Samme tabell som tabell 1-1: MÅL

For å automatisk kunne detektere fartøy har det blitt brukt maskinlæringsmodeller for objektdeteksjon på RPI enheten. Alle modellene som har blitt testet gjennom prosjektet klarer å detektere så å si alle fartøy som kommer i kamerabildet, derimot med ulik grad av sikkerhet og antall deteksjoner. Resultatene fra de forskjellige modellene variasjon i sannsynlighet og FPS (se tabell 4-1). Den trente modellen ssd_mobilenet_v2 FPN-lite og effficientDet_lite2 har lavest FPS, henholdsvis 2,0 og 2,4. Det kan på den ene siden diskuteres for at FPS ikke har så stor innvirkning for deteksjon, da fartøyene som passerer er i kamerabildet i en lang nok periode til at det ikke er avgjørende med høy FPS for å kunne rekke å detektere dem. Men på en annen side viste det seg gjennom testene at når fartøyene holdt en høy hastighet, ble deteksjonsboksene rundt fartøyene mer unøyaktig og det ble gjort færre deteksjoner med modellene som hadde lav FPS. Dette kan skyldes flere faktorer, for eksempel at båter som kjører fort forstyrrer linjene modellen søker etter i bildet. Som oppgaven har diskutert tidligere i avsnitt 2.1.4 er det avgjørende for gode datasett at annoteringsdataen er nøyaktig, og dette understøtter at modellene med lav FPS har en svakhet med hensyn til boksen som settes rundt fartøyet.

Videre har ssd_mobilenet_v2 FPN-lite og effficientDet_lite2 høy sannsynlighet på deteksjonene som blir gjort. I prosjektet lagres alle deteksjoner med sannsynlighet over 50% for å få tatt mange bilder, og samtidig opprettholde en viss terskel for at det skal lagres bilder. Modellen efficientDet_lite0 er den modellen som har minst nevralt nettverk og lavest mAP, noe som også vises gjennom testene ved å være den modellen som hadde lavest sannsynlighet på deteksjonene. Denne modellen hadde også sporadisk unøyaktige annoteringer, selv om den hadde høyest FPS. Dette kan skyldes at modellen rett og slett ikke har et dypt nok nettverk til å dekke kompleksiteten i deteksjonene.

Det kan argumenteres for at ved å bruke modellen efficientDet_lite1, så får man en balansegang mellom deteksjonssannsynlighet og FPS, samt at modellen har ganske nøyaktige annoteringer. Den videretrente ssd_mobilenet_v2 FPN-lite modellen derimot også gode deteksjoner, men vesentlig høyere sannsynlighet. Dette kan skyldes at ssd_mobilenet_v2 FPN-lite er trent på bilder tatt av systemet i miljøet den er testet, noe som også understøtter at dette kan være en god modell til å samle datasett. Stilles derimot spørsmålet om modellvalg opp mot systemets krav om å være en mobil plattform som skal kunne settes ut etter brukers behov, kan det med fordel diskuteres om efficientDet_lite modellene brukes istedenfor en trent modell. EfficientDet_lite modellene fungerte i dette miljøet, og det er plausibelt å anta at disse modellene vil fungere like bra andre steder da de ikke er trent i miljøet testene ble gjort. Det samme kan ikke garanteres for den trente

modellen. På en side strider det også med konseptet å trene en modell til å kjenne igjen båter, da systemet faktisk skal brukes til å samle inn datasett til trening. Videre for dette prosjektet har det derimot vært viktig å likevel trene og teste denne modellen, da det bekrefter at bildene som systemet tar faktisk er mulig å bruke til å trene en god modell.

For å kontinuerlig innhente AIS-data var det nødvendig med tre hovedkomponenter: en kilde som leverte AIS-data, et system som kunne lytte til denne kilden, og et system for å innhente data innenfor et spesifikt geografisk område. I vår løsning ble Kystverkets åpne server benyttet som datakilde, Apache Kafka ble implementert for kontinuerlig avlytting av serveren, og et script med en GPS-ramme ble brukt til å filtrere dataene.

Under de initiale testene ble det konkludert med at Kafka var et brukervennlig og effektivt system for dataoverføring. Kafka var et godt valg for oppgaven, da dette kunne kjøres alene i bakgrunnen og ikke krevde noe form for oppfølging av operatør. Kafka har også et stort potensial for skalerbarhet, og legger til rett for å utvide systemet med flere sensorer. Imidlertid ble det også avdekket at Kystverkets åpne server hadde en relativt lav oppdateringsfrekvens for hver enkelt båt, noe som påvirket nøyaktigheten og aktualiteten til systemet. Som et resultat ble det sendt en søknad til Kystverket om tilgang til den utvidede serveren, som besitter en større datamengde, med hyppigere fartøysoppdateringer. Denne søknaden ble godkjent, men grunnet utfordringer knyttet til anskaffelse av en statisk offentlig IP-adresse for å begrense tilgangen til bare godkjente parter, måtte likevel den opprinnelige åpne serveren bli benyttet.

Scriptet som ble utviklet for å avgrense posisjonsdataen fungerte som forventet. Likevel var selve avgrensningsområdet en utfordring å definere. Etter flere tester ble det konkludert med at systemets avgrensning skulle baseres på kameraets synsfelt (FOV) og den maksimale avstanden som RPI'en kunne oppnå deteksjoner på. Dette ledet til en avveining: På den ene siden, hvis datainnsamlingsområdet ble utvidet utover FOV-en til RPI-en, kunne flere irrevante AIS-meldinger lagres. På den andre siden, hvis datainnsamlingsområdet ble begrenset langs kanten av kameraets synsfelt, kunne det være en risiko for å gå glipp av relevante AIS-meldinger fra fartøy som befant seg nær kanten. Grunnet den lave oppdateringsfrekvensen til AIS-meldingene, konkluderte vi med at GPS-rammen måtte utvides utover FOV-en til kamerabildet.

Ulempen med å sette en statisk GPS-ramme er dens mangel på fleksibilitet. Avgrensningen plottes manuelt, og er dermed sårbar ved fysiske endringer på kamerabildet. Ved å flytte kameraet til en annen posisjon eller endre orienteringen til kameraet, vil man måtte

rekalibrere avgrensingene. Noe som resulterer i at dette blir en del av oppstartsprosessen ved oppsett av systemet på ønsket plass. Det er flere måter å gjøre systemet dynamisk. På en side kan man installere og ta i bruk GPS og kompass, for å bestemme kameraets posisjon og orientering. På en annen side kan man bruke systemet slik det er nå ved å kalibrere kameraet etter første passering med AIS, dette krever riktig nok en AIS-innhenting som er hyppig og nøyaktig for å bestemme posisjoner i kamerabildet.

AIS-meldingene vi mottok inneholdt store mengder informasjon. Dette resulterte i en avveining om hva som var nødvendig å ha med, og valget som ble tatt var å kun inkludere systemnødvendig informasjon. Fordelen med den kraftige filtreringen av AIS-meldinger er at det øker effektiviteten til systemet ved å redusere mengden data som må prosesseres og lagres. Ulempen ved den kraftige filtreringen er at man begrenser tilgjengelig informasjon om observerte fartøy, som potensielt kunne vært nyttig til forskjellige måter å trene på i etterkant. Likevel, er filtreringen noe som enkelt kan endres på i koden, slik at potensiell videre implementering blir enkel å foreta. Det bør derfor vurderes hvilken informasjon som er verdt å ta vare på før systemet settes i drift i det på hvert enkelt område.

Det andre målet som ble satt for oppgaven var at dataen som ble innhentet fra de forskjellige kildene, skulle håndteres videre av et program og sammenlignes. Hvis sammenligningen av deteksjonsdataen og AIS-informasjonene samsvarte, skulle all informasjonen lagres sammen. For å sammenligne informasjonen, ble det bestemt å sammenligne tidspunkt og posisjonen fra deteksjonsdataen opp mot den innhentete AIS-informasjonen. Som testingen tar for seg i kapittel 4.3.1 ble bufferen for både tid og posisjon justert opp. Det kan diskuteres for at det er flere svakheter med denne løsningen. Grunnet den lave oppdateringsfrekvensen til AIS dataen og at tidsbufferen ble skrudd opp til to minutter, medførte dette at det potensielt kunne være flere AIS meldinger som ble linket opp med feil deteksjonsbilder. På den ene siden er det klart at jo større tidsrammen ble, jo mindre nøyaktig blir sammenligningen. På den andre siden kan det argumenteres for at tidsaspektet i sammenligningen er mindre betydelig, hvis AIS-meldingene kommer inn med høyere frekvens. Årsaken er at hvis det hadde blitt sendt ut meldinger hvert tredje sekund, hadde det vært mange båter som kunne befunnet seg innenfor en rimelig tidsbuffer uansett, og at det dermed er posisjonen som er det viktigste i sammenligningen.

Alle deteksjoner fra kamera får samme posisjonskoordinater som sammenlikningsgrunnlag, noe som i seg selv ikke er nøyaktig, og med denne posisjonsbufferen rundt

faktisk posisjon øker også sannsynligheten for feil. En løsning på dette kunne vært å implementere en måte å kalibrere kameraet på, slik at alle fartøy får egne koordinater ut ifra plassering i kamerabildet. På en annen side vil nok dette også gi liten effekt når kameraet burde plasseres på en slik måte at området er begrenset, da det ikke vil gi store endring fra en statisk gitt posisjon.

Et spørsmål som er viktig å stille seg er hva som skjer når kameraet har to båter i bildet samtidig. Pr. nå klarer modellene å lagre bilder med deteksjonsrammer rundt alle fartøy som er i bildet. Men en svakhet er at sånn som programmet er nå, vil dette allikevel bli behandlet som én deteksjon. Dette betyr videre at det er båten som har tilhørende AIS-informasjon nærmest den statiske posisjonen, som blir deteksjonen sammen med dataen. Dette er en svakhet, da man ikke skiller på alle deteksjonene, og ikke minst fordi fartøy som kanskje ikke benytter AIS kan bli detektert sammen med et annet skip. Dette er ugunstig da båter uten AIS-utsending kan være av større interesse, noe som blir diskutert i kapittel 5.2.

Det tredje målet som ble satt for prosjektet var at datapakker med relevant informasjon om en deteksjon skal lagres i en database som er tilgjengelig for relevante andre. For å kunne lagre ønsket informasjon i en database valgte vi å benytte oss av SQLite. Databasen ble opprettet lokal på den bærbare computeren, og det ble implementert en kode for å sende behandlet data til databasene. Strukturen i databasen, som er illustrert i figur 4-19, ble valg basert på hvilke parametere vi anså som relevante for deteksjonen. Databasens tilgjengelighet for relevante andre, er til nå noe begrenset, da dataene kun lagres lokalt på den bærbare computeren. Dataoverføring må dermed gjøres manuelt, enten fysisk eller trådløst. På den ene siden er SQLite en ressursvennlig database, da den er innebygd og serverløs, noe som gjør det ideelt for applikasjonene som kjører simultant på den bærbare computeren. På den andre siden har databasen begrenset tilgjengelighet for relevante andre, da dataen kun lagres lokalt på den bærbare computeren. Dette kan gjøre det utfordrende å dele og samarbeide med dataen. Likevel er det fullt mulig å videreutvikle systemet på denne fronten, ved å eksempelvis ta i bruk en skyløsning som vil være tilgjengelig for andre. En internettilgang er enn så lenge uansett nødvendig for å få tilgang til AIS-data.

Det fjerde målet som ble satt for prosjektet var at det skal være mulig for en eventuell operatør å se når nye datapakker kommer inn i databasen i sanntid, samt å overvåke sys-

temets drift. For å holde systemet oversiktlig for operatøren, implementerte vi bekreftelsesmeldinger i terminalvinduet under dataoverføring og sammenlikning, som illustrert på figur 4-12, 4-15 og 4-17. Som nevnt tidligere i oppgaven, informerer også figur 4-12 om AIS-innhentingens systemstatus. Likevel, når RPI'en kjører deteksjonsprogrammet uten en skjermoutput, er det ingen form for status på at deteksjonsprogrammet kjører før en deteksjon overføres. Overførselen vises som illustrert på figur 4-15. En rask implementering som kan forbedre operatørens oversikt er dermed at det overføres en bekreftelsesmelding over Falsk-serveren da deteksjonsprogrammet kjører, og når det stopper.

DB Browser for SQLite ble tatt i bruk for å gi operatøren en visuell oversikt over databasen, samt muligheten til å interagere. Visualiseringen til databasen er illustrert på figur 4-20. Fordelen med bruken av DB4S som et visualiseringsverktøy er brukervennlighet, da den gir et grafisk grensesnitt for å utforske, redigere og visualisere dataene, uten behovet for å skrive komplekse SQL-spørringer manuelt. I tillegg, er det en gratis åpen kildekode, noe som gjør det til et kostnadseffektivt alternativ for visualisering.

Haken ved å benytte DB4S er den noe begrensede visualiseringen, noe som eksempelvis vises når det ikke er mulig å visuelt vise bildene i databasen. Likevel vil vår løsning fungerer fint ved å referere til stien hvor bildene er lagret. En alternativ løsning er å lagre bildene som Binary Large Object (BLOB) i databasen, som vil si at man lagrer binærdataen til bildene. (Burchfiel, 2022)

Steget videre for visualiseringen hadde potensielt vært å benytte seg av HTML for videre visualisering av systemet. Dette hadde muliggjort trådløs overvåking for en operatør, slik at vedkommende ikke har behov for å være fysisk ved siden noen av enhetene, men kun benytte seg av en ekstern enhet tilkoblet via internett.

5.2 Potensialet i en sjømilitær kontekst

Som nevnt innledningsvis, skriver FFI følgende på sin nettside: «Oppdatert overvåking står sentralt i militære operasjoner, på hele skalaen fra fred og kriser til full krig. […] » (FFI, u.d.). Dette belyser relevansen for utvikling av maritim overvåkning. Systemet tilrettelegger for nettopp videreutvikling av maritime overvåkingssystemer, ved å etablere datasett for trening av maskinlæringsapplikasjoner i spesifikke miljø.

Med evnen til å identifisere fartøy som mangler AIS-data, kan systemet bidra til å oppdage skip av interesse som potensielt kan være en sikkerhetstrussel eller involvert i ulovlige aktiviteter, slik som smugling eller ulovlig fiske. På denne måten utvider systemet ikke bare måten å overvåke fartøy med AIS, men gir også muligheter for å forbedre overvåkingen av fartøy uten AIS.

Ved å gjøre databasen online, blir det mulig for eksterne operatører å monitorere sanntidsdeteksjoner. Systemet kan dermed tilby betydelig forbedret situasjonsbevissthet langs kysten med et minimum av kostnader tilknyttet infrastrukturen. Ved å automatisk detektere og samle data om fartøy, vil militære operatører ha oppdatert og nøyaktig informasjon om skipstrafikk og mulige trusler i sine territorialfarvann.

Ved å bruke maskinlæringsmodeller kan overvåkning gjøres mer effektivt ved å redusere behovet for mannskap og ressurser til sjøs. Dette kan igjen øke dekningsområdet og frekvensen av overvåking, samtidig som det frigjør ressurser til å løse andre oppdrag.

6 Konklusjon

I dette kapittelet skal først prosjektets måloppnåelse konkluderes. Dette blir gjort ved å sette prosjektets design opp mot problemstillingen og kravene dette spørsmålet satt for systemet. Videre skal avslutningen ta for seg våre anbefalinger til forbedringer og videre arbeid.

I vår løsning på problemstillingen har det blitt brukt maskinlæringsmodeller for objektdeteksjon, sammen med innhenting av AIS-data. Ved å benytte en Raspberry Pi er mobilt plasserbar i utsatte områder, for å operere autonomt på lokasjoner av interesse. Resultatene viser at systemet er i stand til å samle inn både deteksjonsinformasjon og AIS-meldinger, samt korrelere dataene for å skape datasett som inneholder bilder, annoteringsdata og informasjon om fartøyet, så vidt fartøyet har AIS. Dessuten er systemet i stand til å samle bilder av fartøy uten AIS.

Den trente modellen er et bevis på at dataen som samles inn er av god nok kvalitet til å trene en fungerende maskinlæringsmodell. Resultatene tilsier også at den trente modellen har en god evne til å identifisere fartøy med høy grad av presisjon, noe som understøtter systemets potensiale som en datakilde for fremtidig modelltrening. Det kan derfor konkluderes med at systemet og konseptet i sin helhet er en mulig løsning på problemstillingen. Modellens prestasjon i forhold til FPS og i scenarioer med høyhastighetsobjekter må vurderes i praksis.

Løsningen i dette prosjektet er bygget på et fundament av lett tilgjengelig maskinvare og åpenkildekode. Det kan derfor konkluderes med at systemet har stort potensiale for oppgraderinger og forbedringer. Videre bekrefter derimot dette også at det ikke skal så store ressurser til for å oppnå ønsket effekt innenfor dette feltet av maritim overvåkning. Dette må sies å være meget relevant, sett i sammenheng med at budsjett spiller en stor rolle i gjennomføringen av slike prosjekter.

Systemet tilfredsstiller altså kravene om mobilitet og funksjonalitet for datainnsamling, og etablerer et solid fundament for fremtidige forbedringer. Systemets evne til å hente og lagre data fra et ønsket område er av tilfredsstillende kvalitet, og bekrefter at prosjektets løsning kan være med å støtte oppunder fremtidige maskinlæringsinitiativer som blir igangsatt innenfor den maritime sektor.

For å avslutte oppgaven ønsker vi å komme med våre anbefalinger til utbedring av konseptet, og videre arbeid som kan vurderes. Utbedringen kan i all hovedsak deles inn i to kategorier:

- Hva som kan gjøres for å forbedre komponentene i det nåværende systemet.
- Hva som kan legges til for å videreutvikle konseptet.

Kodene fungerer på dette tidspunktet på en måte som gjør at man må sette en statisk koordinatposisjon ved utplassering av systemet. For å gjøre systemet enda mer autonomt kan man plassere en GPS-mottaker sammen med systemet, kameraet vil da potensielt kunne oppdatere sin egen posisjon kontinuerlig ved flytting. Videre burde det utarbeides en måte for nøyaktig å kunne finne posisjonene til båtene. Dette kan muligens gjøres ved å kalibrere kameraet ut ifra GPS posisjonen og omgivelsene i kamerabildet, også kalt georeferencing. En annen spennende løsning på dette som potensielt kan føre til at systemet blir enda mer autonomisert, er å se på muligheten for å implementere en laseravstandsmåler. Dette kan muligens brukes til å finne posisjonen til fartøyene gjennom kameraets GPS-posisjonen, relativ peiling og den målte avstanden. Videre kan man også se på muligheten for å bruke en AIS-mottaker som benytter VHF-radiofrekvenser, da slipper systemet å belage seg på utsendelse fra kystverkets servere, noe som gjør systemet uavhengig av internett.

Slik kameraet er satt opp nå, klarer ikke systemet å få tatt bilder når det er mørkt eller når det er dårlig vær/mye tåke. Det kan vurderes å bruke en form for nattlinse for å løse dette, men fortsatt løses ikke værutfordringene. Vårt anbefalte tiltak er å ta inn et infrarødt (IR) kamera som kan oppdage varmesignaturen til fartøy, noe som også potensielt har kapasitet til å kunne ta mer enn gode nok bilder til bruk for maskinlæringsmodeller.

For å utvikle konseptet videre kan det legges til flere komponenter både for å bekrefte at det er et fartøy til stede i bildet, og for å kunne samle inn enda mer data som kan være av interesse for andre typer maskinlæringsmodeller enn det som er brukt i dette prosjektet. Ved å bruke en radar sammen med det nåværende systemet, kan man kryssvalidere at det er en deteksjon mellom radaren og kameraet, samtidig som man også kan lagre bilder av radarsignaturen til fartøyene. Ved å kartlegge radarsignaturene kan dette potensielt brukes til å utvikle eksisterende radarteknologi ombord på skip, og mulig bidra til militær oppdragsløsning i maritim sektor. Videre kan man også se på muligheten for å implementere en hydrofon som kan gjøre opptak av lydsignaturene ved deteksjoner. Hvis dette lagres

sammen med informasjonen i datasettene, kan man potensielt kunne trene modeller til å kjenne igjen disse lydsignaturene.

Avslutningsvis anbefaler vi også at det blir gjennomført en litteraturstudie om hvordan maritim overvåkning blir gjennomført i dag, hvordan utviklingen innenfor fagfeltet har vært, og ser ut til å bli i fremtiden. Dette kan bidra til å videreutvikle konseptet som er lagt til grunnlag for denne oppgaven, og samtidig gi fundament for nye løsninger på problemstillinger denne oppgaven har utredet.

7 Bibliografi

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen , Z., Citro, C., . . . Jozefowicz, R. (2015). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. Google Research.* Hentet fra Tensorflow.org.

Allan, A. (2018). *Benchmarking Tensorflow and Tensorflow Lite on the raspberry pi.* Hentet fra https://www.hackster.io: https://www.hackster.io/news/benchmarking-tensorflow-and-tensorflow-lite-on-the-raspberry-pi-43f51b796796

Baheti, P. (2021, Oktober 1). *Supervised and Unsupervised Learning [Differences & Examples]*. Hentet fra https://www.v7labs.com: https://www.v7labs.com/blog/supervised-vs-unsupervised-learning

Boesch, G. (2022). *TensorFlow Lite – Real-Time Computer Vision on Edge Devices.* Hentet fra https://viso.ai/: https://viso.ai/edge-ai/tensorflow-lite/

Burchfiel, A. (2022, 04 05). *What is a BLOB (Binary Large Object)? Can it be Tokenized?* Hentet fra Cloud security alliance: https://cloudsecurityalliance.org/blog/2022/05/04/what-is-a-blob-binary-large-object-can-it-be-tokenized/

DB Browser for SQLite. (2021, Mai 18). Hentet fra https://sqlitebrowser.org/: https://sqlitebrowser.org/

Dewang, N. (2023, August 31). *ML | Underfitting and Overfitting*. Hentet fra https://www.geeksforgeeks.org: https://www.geeksforgeeks.org/underfitting-and-overfitting-in-machine-learning/

Farnell. (2023). *RASPBERRY-PI RPI4-MODBP-8GB.* Hentet 2023 fra https://no.farnell.com: https://no.farnell.com/raspberry-pi/rpi4-modbp-8gb/raspberry-pi-4-model-b-cortex/dp/3369503

FFI. (u.d.). *Overvåking*. Hentet fra https://www.ffi.no: https://www.ffi.no/forskning/tema/radar

Flask. (2023). *API.* Hentet 11 23, 2023 fra Flask: https://flask.palletsprojects.com/en/3.0.x/api/

Google. (2023, 10 12). *Google Maps*. Hentet fra Google: https://www.google.com/maps/place/Bergen/@60.4060006,5.2697794,13.52z/d

ata=!4m6!3m5!1s0x46390d4966767d77:0x9e42a03eb4de0a08!8m2!3d60.3912628!4d5.3220544!16zL20vMGZtN3M?entry=ttu

Hollup, O., & Røsand, M. (2022). *Implementering av objektdeteksjon på maritim dronesverm.* Bergen: Bacheloroppgave, FHS Sjøkrigsskolen.

Horten, E., Ytterstad, S., & Rugahiye, W. (2021). *FARTØYSOVERVÅKNING.* Bergen: Bacheloroppgave, FHS Sjøkrigsskolen.

Hui, J. (2018, 03 27). *Understanding Feature Pyramid Networks for object detection (FPN)*. Hentet fra Medium: https://jonathan-hui.medium.com/understanding-feature-pyramid-networks-for-object-detection-fpn-45b227b9106c

IBM. (2023, 11 15). *What are convolutional neural networks?* Hentet fra IBM: https://www.ibm.com/topics/convolutional-neural-networks

IBM. (2023). *What are neural networks*. Hentet fra https://www.ibm.com: https://www.ibm.com/topics/neural-networks

International Maritime Organization. (u.d.). *AIS transponders*. Hentet 11 29, 2023 fra IMO: https://www.imo.org/en/OurWork/Safety/Pages/AIS.aspx

Jakobsen, Ø., Paramanathan, J., & Sommerfeldt, S. (2023). *Hva er Kafka og hvorfor har det blitt så populært*. Hentet 11 26, 2023 fra Bouvet: https://www.bouvet.no/podcasts/hva-er-kafka-og-hvorfor-har-det-blitt-sa-populaert

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

Juras, E. (2023, Januar 28). *TensorFlow Lite Object Detection API in Colab*. Hentet fra https://colab.research.google.com: https://colab.research.google.com/github/EdjeElectronics/TensorFlow-Lite-Object-Detection-on-Android-and-Raspberry-Pi/blob/master/Train_TFLite2_Object_Detction_Model.ipynb

Juras, E., Tielens, J., Trax, & Hong, F. (2022, Mars 1). *TensorFlow-Lite-Object-Detection-on-Android-and-Raspberry-Pi*. Hentet fra https://github.com: https://github.com/EdjeElectronics/TensorFlow-Lite-Object-Detection-on-Android-and-Raspberry-Pi

Kafka. (2023, 11 13). *Kafka*. Hentet fra Documentation: https://kafka.apache.org/documentation/#gettingStarted

Kaggle. (2020, Oktober 6). *ssd_mobilenet_v2*. Hentet fra https://www.kaggle.com: https://www.kaggle.com/models/tensorflow/ssd-mobilenet-v2/frameworks/tensorFlow2/variations/fpnlite-320x320/versions/1?tfhub-redirect=true

Krishnamurthy, B. (2022, 10 28). *An introducton to the ReLU Activation Function*. Hentet fra Builtin: https://builtin.com/machine-learning/relu-activation-function

Kystverket. (2019). *Hva er AIS?* Hentet fra Kystverket.no: https://havbase.kystverket.no/havbase_report/doc/AIS.pdf

Kystverket. (2020, Oktober 22). *Vil styrke overvåking av skipstrafikk*. Hentet November 6, 2023 fra Kystverket.no: https://www.kystverket.no/nyheter/2020/vil-styrke-overvaking-av-skipstrafikk

Kystverket. (2023, 11 06). *Fakta om AIS*. Hentet fra kystverket.no: https://www.kystverket.no/navigasjonstjenester/ais/ais-artikkelside/

Kystverket. (2023, 11 14). *Kystverket*. Hentet fra Tilgang på AIS-data: https://www.kystverket.no/navigasjonstjenester/ais/tilgang-pa-ais-data/

Kystverkets hovedkontor, Sjøsikkerhetsavdelingen. (2019, Mars). *Kystverket.* Hentet fra Retningslinjer for bruk av AIS-navigasjonsinnretninger: https://www.kystverket.no/globalassets/bildervedleggfaktabokser/sjovegen/fyr-lykter-og-sjomerker/retningslinjer-for-bruk-av-aisnavigasjonsinnretninger-2019-.pdf

Løkke, E. (2023, November 7). *Overvåkning*. Hentet fra https://civita.no: https://civita.no/politisk-ordbok/overvakning/

Maskinlæringsdefinisjon i detalj. (u.d.). Hentet fra www.sap.com: https://www.sap.com/norway/products/artificial-intelligence/what-is-machine-learning.html

Mishra, M. (2020, 08 26). *Convolutional Neural Networks, Explained*. Hentet fra Medium: https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939

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

Papaemmanouil, P. (2022, Juni 18). *Classical programming vs. Machine Learning in plain English.* Hentet fra https://medium.com: https://medium.com/codex/classical-programming-vs-machine-learning-in-plain-english-3f39c56673d9

Peng, J., Jury, E., Dönnes, P., & Ciurtin, C. (2021, September). *Machine Learning Techniques for Personalised Medicine Approaches in Immune-Mediated Chronic Inflammatory Diseases: Applications and Challenges.* Hentet fra https://www.researchgate.net/: https://www.researchgate.net/publication/354960266_Machine_Learning_Techniques_for_Personalised_Medicine_Approaches_in_Immune-Mediated_Chronic_Inflammatory_Diseases_Applications_and_Challenges

Platon. (2023, 08 1). *En omfattende guide til UNET-arkitektur | Mestring av bildesegmentering* . Hentet fra PLATO AI: https://zephyrnet.com/no/en-omfattende-guide-til-unet-arkitektur-som-mestrer-bildesegmentering/

PyPi. (2021, Oktober 10). *labelImg 1.8.6*. Hentet fra https://pypi.org: https://pypi.org/project/labelImg/

Python Basics. (2023, 11 13). *Python Basics*. Hentet fra Python Tutorial: https://python-basics.org/what-is-flask-python/

Qlink. (2023, 11 15). *Apache Kafka*. Hentet fra qlink.com: https://www.qlik.com/us/streaming-data/apache-kafka

Raspberry Pi. (2023). *Raspberry Pi High Quality Camera*. Hentet fra https://www.raspberrypi.com: https://www.raspberrypi.com/products/raspberry-pi-high-quality-camera/

Raspberry Pi. (2023). *Raspberry Pi OS*. Hentet fra https://www.raspberrypi.com: https://www.raspberrypi.com/software/

Ruiz Brothers. (2020, Juni 5). *Raspberry Pi HQ Camera Case*. Hentet fra https://learn.adafruit.com: https://learn.adafruit.com/raspberry-pi-hq-camera-case/3d-printing

Soulaimaneyh. (2023, August 13). *Exploring the Fundamental Principles of Distributed Systems*. Hentet fra https://medium.com: https://medium.com/@soulaimaneyh/exploring-the-fundamental-principles-of-distributed-systems-970c285a77b5

SQLite. (2022, 12 16). *Appropriate Uses For SQLite*. Hentet fra SQLite.org: https://www.sqlite.org/whentouse.html

Supervised Machine Learning. (u.d.). Hentet 2023 fra https://www.javatpoint.com: https://www.javatpoint.com/supervised-machine-learning

Tan, M., Pang, R., & Le, Q. (2020, Juli 27). *EfficientDet: Scalable and Efficient Object Detection*. Hentet fra https://arxiv.org: https://arxiv.org/abs/1911.09070

Tan, R. (2019, Mars 24). *Breaking Down Mean Average Precision (mAP)*. Hentet fra https://towardsdatascience.com: https://towardsdatascience.com/breaking-down-mean-average-precision-map-ae462f623a52

Tensorflow. (2021). *TensorFlow Lite Python object detection example with Raspberry Pi*. Hentet fra https://github.com: https://github.com/tensorflow/examples/tree/master/lite/examples/object_detection/raspberry_pi

Tensorflow. (2023). Hentet fra Tensorflow.org: https://www.tensorflow.org/

Tensorflow. (2023, Mai 5). *Object Detection with TensorFlow Lite Model Maker*. Hentet fra https://www.tensorflow.org: https://www.tensorflow.org/lite/models/modify/model_maker/object_detection

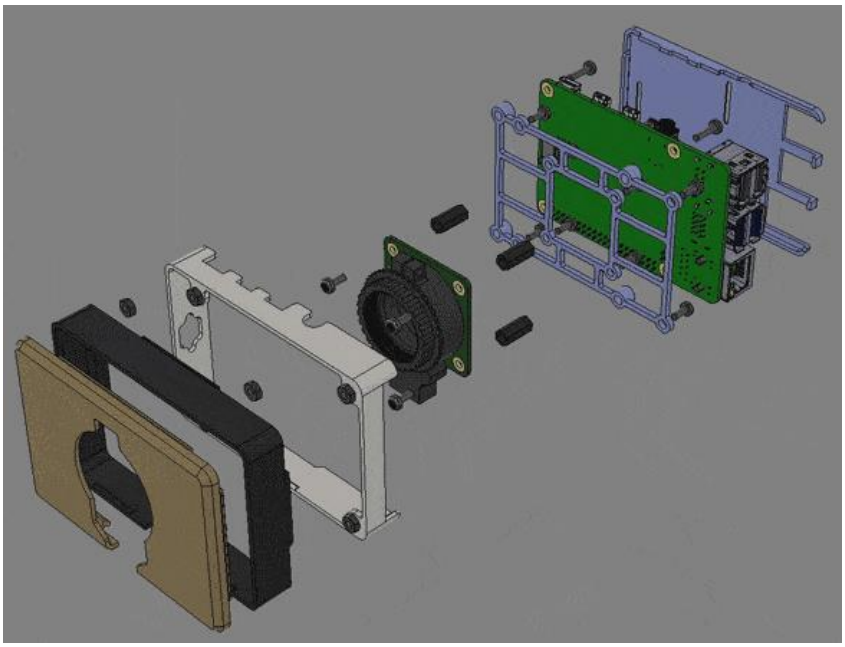
Tensorflow. (2023, Oktober 27). *Transfer learning and fine-tuning*. Hentet fra https://www.tensorflow.org: https://www.tensorflow.org/tutorials/images/transfer_learning

Tidemann, A., & Elster, A. (2023, 26 Juli). *Maskinlæring*. Hentet fra SNL.no: https://snl.no/maskinl%C3%A6ring

Tielens, J., Trax, Hong, F., & Evan. (2022, Mars 1). *TensorFlow-Lite-Object-Detection-on-Android-and-Raspberry-Pi*. Hentet fra https://github.com: https://github.com/EdjeElectronics/TensorFlow-Lite-Object-Detection-on-Android-and-Raspberry-Pi

UNSD_MM. (2020, 05 05). *Overview of AIS dataset*. Hentet fra UN Statistics Wiki: https://unstats.un.org/wiki/display/AIS/Overview+of+AIS+dataset

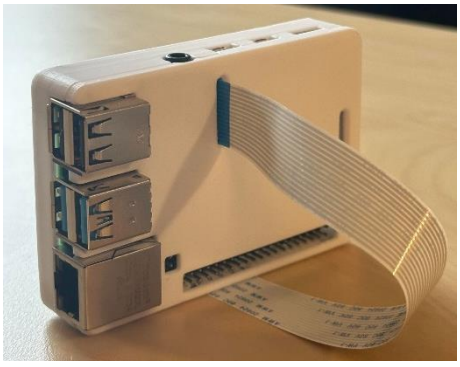
Vedlegg A - RPI/HQcamera sammensetning



Kan lastes ned og pintes fra:

[3D Printing | Raspberry Pi HQ Camera Case | Adafruit Learning System]

Oppbygningen til kamerahuset som er brukt i oppgaven, ytterligere forklaringer og hvordan man kan sette dette sammen finnes på lenken.
Flere bilder av oppgavens oppsett:



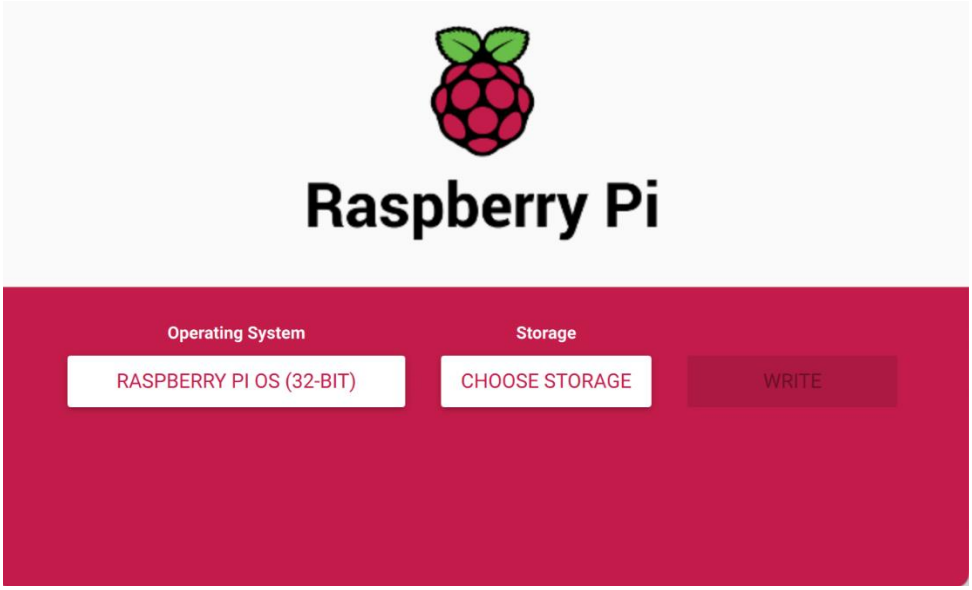
Vedlegg B – Operativsystem på RPI & Computer

Raspberry Pi

Raspberry Pi bruker et SD-kort til å lagre alt som er på enheten. Før man starter opp RPI'en kan man derfor laste ned et operativsystem på SD-kortet som skal brukes

For å laste ned operativsystemet, kan man laste ned er program som heter «Raspberry Pi Imager» fra Raspberry Pi nettsiden på en annen enhet: Raspberry Pi OS – Raspberry Pi

Når programmet er lastet ned kobler man SD-kortet til enheten åpner programmet. Programmet vil da se slik ut:



I dette prosjektet det det brukt **Raspberry Pi OS 11 – Bullseye**

Computer

Computeren som benyttes er en Lenovo Thinkpad, som kjører **Ubuntu 22.04.3 LTS**

Installasjonslink: Download Ubuntu Desktop | Download | Ubuntu

Anbefaling

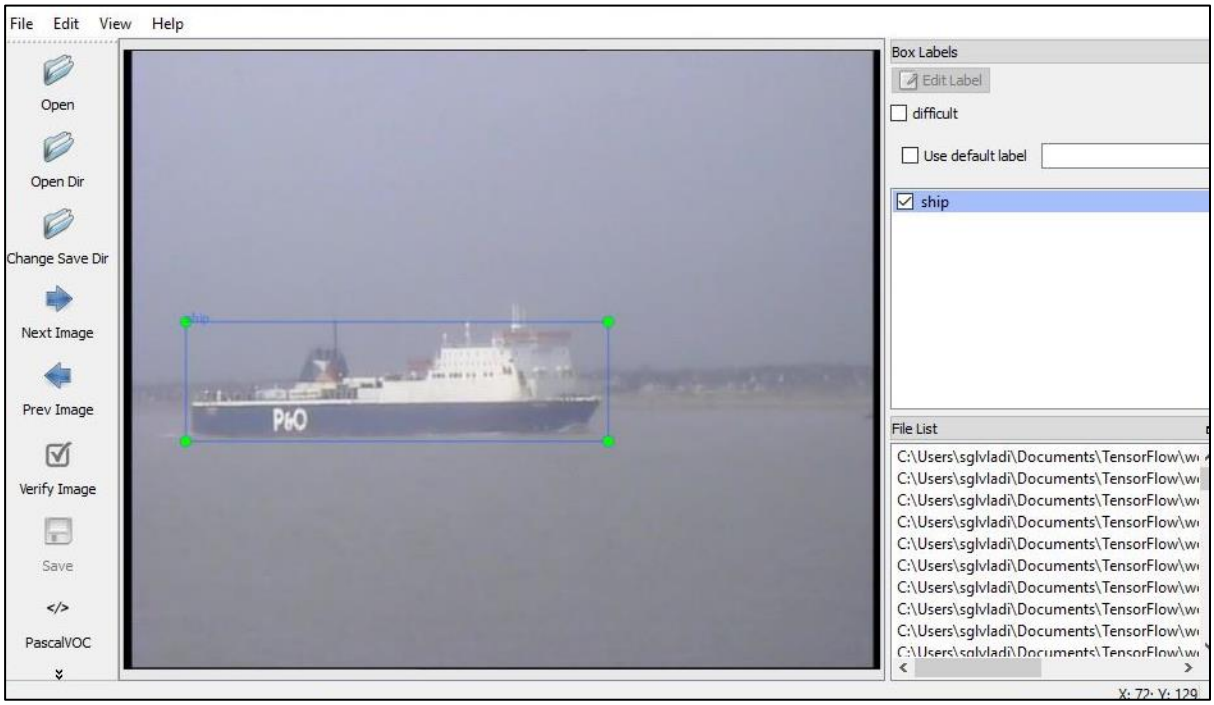
Vi anbefaler derfor å velge de samme operativsystemene, for å garatere at det ikke vil oppstå komplikasjoner med endringer som er gjort fra eldre eller med ny versjoner.

Vedlegg C – LabelImg

LabelImg er programmet som er brukt til å annotere bildene i datasettet som er brukt til å trene ssd_mobilenet_v2 FPN-lite modellen. Programmet er brukervennlig og oppdaterer automatisk bildemappen med XML-filer tilhørende bildene.

Programmet kan lastet ned på forskjellige systemer, og programmet finner man på denne nettsiden: https://pypi.org/project/labelImg

Programmet ser slik ut ved bruk:



Vedlegg D – Kildekode lagret i GitHub

Introduksjon til GitHub

GitHub er en populær webbasert plattform brukt for versjonskontroll av kildekode. Den tilbyr både enkel lagring og tilgang til koden, hvor det er enkelt å laste den den nyeste versjonen av koden og samtidig enkelt å laste ned fra en ekstern enhet.

Nedlasting av Kildekode

For å laste ned kildekode fra GitHub, må brukerne først installere Git på deres Raspberry Pi. Dette kan gjøres ved å åpne Terminalen og skrive inn følgende kommando:

sudo apt install git

Prosjektlenker

For dette prosjektet er kildekoden delt inn i tre hoveddeler:

1. Deteksjonsprogram (RPI):
   https://github.com/MBachJ/Detection
   https://github.com/MBachJ/Detection2
2. AIS-innhenting (Computer):
   https://github.com/MBachJ/Computer-Programs/tree/main/Innhenting_Konvertering
3. Sammenlikning & Lagring (Computer):
   https://github.com/MBachJ/Computer-Programs/tree/main/Sammnefatning_Lagring

Dersom det er noen spørsmål til programvaren – Ta kontakt!

Vedlegg E – Installasjon av objektdeteksjon på RPI

Som beskrevet i oppgaven er det et program for å kjøre efficientDet_lite modellene og et program for å kjøre ssd_mobilenet_v2 FPN-lite modellen.

For å bruke programmene er det anbefalt å bruke et virtuelt miljø, og i dette prosjektet er det brukt to forskjellige miljøer for hver av programmene. Vi anbefaler å gjøre dette hvis programmene fra oppgaven skal bli brukt videre. Et virtuelt miljø tillater at hvert prosjekt å ha sine egne avhengigheter, uavhengig av andre prosjekter. Dette betyr at man kan ha ulike versjoner av biblioteker og pakker installert, og det virtuelle miljøet gjør at man unngår konflikter mellom dem.

Installer pakken for å lage virtuelt miljø:

python3 -m pip install –user virtualenv

Opprett en mappe til hvert av programmene:

mkrdir deteksjon

mkrdir deteksjon2

Lag et virtuelt miljø i begge mappene:

python3 -m venv ~/deteksjon/tflite

python3 -m venv ~/deteksjon2/tflite2

tflite og tflite2 blir navnene på miljøene, dette kan du velge selv.

Videre skal dette vedlegget først vise installasjon av programmet som kan kjøre efficient-Det_lite modellene.

Gå til mappen og aktiver miljøet

cd deteksjon

source tflite/bin/activate (her aktiveres miljøet, dette må gjøre hver gang man skal bruke programmene. For å gå ut av miljøet skriver man «deactivate»)

Hent originalprogrammer fra tensorflow, og programmene fra dette prosjektet:

git clone https://github.com/tensorflow/examples.git *(kopierer fra tensorflow)*

cd examples/lite/examples/object_detection/raspberry_pi

git clone https://github.com/MBachJ/Detection *(kopierer programmer fra prosjektet)*

cd raspberry_pi

Installer tensorflow plattformen og nødvendige pakker

sh setup.sh

Start programmet for sending og objektdeteksjon

python3 data_overføring.py & *(& gjør at programmet kjører i bakgrunnen. Sørg for at Flask-serveren kjører på den andre PC'en, og bytt ut IP-adressen i programmet med riktig adresse, og bytt ut stien med stien til deres mappe for lagrete deteksjoner)*

python3 boat_detect.py *(objektdeteksjon er nå i gang, med default efficientDet_lite 0)*

python3 boat_detect.py --model lite-model_efficientdet_lite1_detection_meta-data_1.tflite *(objektdeteksjon med efficientDet_lite 1)*

python3 boat_detect.py --model lite-model_efficientdet_lite2_detection_meta-data_1.tflite *(objektdeteksjon med efficientDet_lite 2)*

Ved feilmelding om biblioteker, prøv:

sudo apt-get install libopenblas-dev

eller

sudo apt-get install libatlas-base-dev

Resten av vedlegget skal ta for seg installasjon og kjøring av programmet som kan kjøre ssd_mobilenet_v2 FPN-lite

Gå til mappen og aktiver miljøet

cd deteksjon2

source tflite2/bin/activate (her aktiveres miljøet, dette må gjøre hver gang man skal bruke programmene. For å gå ut av miljøet skriver man «deactivate»)

Hent programmene

git clone https://github.com/MBachJ/Detection2

cd Fartøys_deteksjon

Installer tensorflow plattformen og nødvendige pakker

sh get_pi_requirements.sh

Start programmet for sending og objektdeteksjon

python3 data_sender.py & *(& gjør at programmet kjører i bakgrunnen. Sørg for at Flask-serveren kjører på den andre PC'en, og bytt ut IP-adressen i programmet med riktig adresse, og bytt ut stien med stien til deres mappe for lagrete deteksjoner)*

python3 TFLite_detection_webcam.py --modeldir=Sample_TFlite_model *(kjører standardmodell av ssd_mobilenet_v2 FPN-lite, som kan kjenne igjen mange objekter)*

python3 TFLite_detection_webcam.py --modeldir=RPIbilder_ssd_mobilenet_v2_fpnlite_320 *(kjører den trente modellen, som kan kjenne igjen båter fra dette prosjektet)*

For enda nøyere installasjons instruksjoner og tips til dette programmet:

Youtube: https://www.youtube.com/watch?v=aimSGOAUI8Y

Vedlegg F – Flask installasjon

Oppdater Systemet

Før du installerer nye pakker, er det en god praksis å oppdatere systemet med følgende kommandoer:

sudo apt-get update

sudo apt-get upgrade

Installer Python og Pip

Flask er bygget på Python, så du må sørge for at Python og pip (Python-pakkeadministratoren) er installert. Bruk følgende kommandoer:

sudo apt-get install python3

sudo apt-get install python3-pip

Installer Flask

Benytt følgende kommando for å laste ned Flask-systemet:

pip3 install Flask

Implementer kildekode

Flask-kode:

https://github.com/MBachJ/Computer-Programs/blob/main/Sammnefatning_Lagring/flask_server.py

Vedlegg G – Apache Kafka installasjon

Forutsetninger

Kafka er skrevet i Java og buker ZooKeeper. Dermed må begge program installeres:

Java:

sudo apt update

sudo apt install default-jdk

Zookeeper:

Last ned Zookeeper fra Apache Zookeeper's offisielle nettside.

Pakk ut, eksempelvis i mappe opt

tar -xvf zookeeper-<version>.tar.gz -C /opt

Opprett en konfigurasjonsfil «zoo.cfg» i Zookeper's conf-mappe

cp /opt/zookeeper/conf/zoo_sample.cfg /opt/zookeeper/conf/zoo.cfg

Start Zookeeper-serveren

/home/zookeeper/bin/zkServer.sh start

Kafka

Last ned Kafka fra Apache Kafka's offisielle nettside.

Pakk ut til en valgt mappe, eksempelvis i opt

tar -xvf kafka_<version>.tgz -C /opt

Start serveren

/opt/kafka/bin/kafka-server-start.sh /opt/kafka/config/server.properties

Implementer kildekode

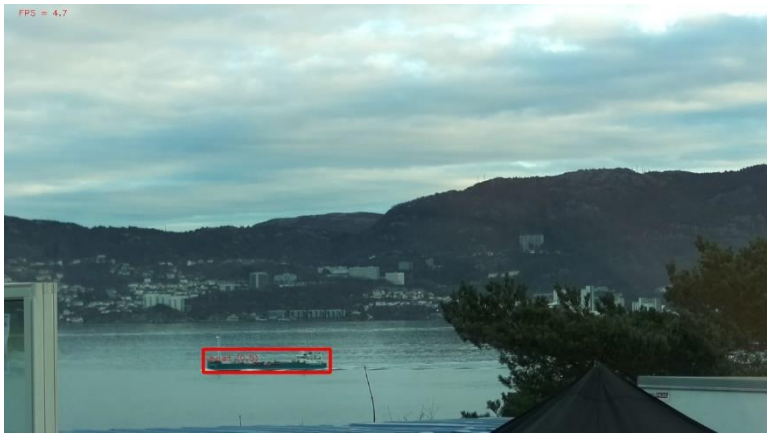
decode_ais.py & run_ais.sh:

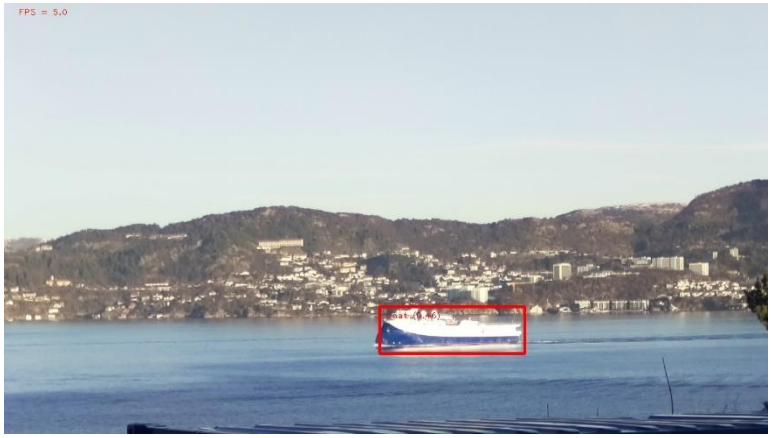
https://github.com/MBachJ/Computer-Programs/tree/main/Innhenting_Konvertering

Vedlegg H - Tall fra sannsynlighetstester

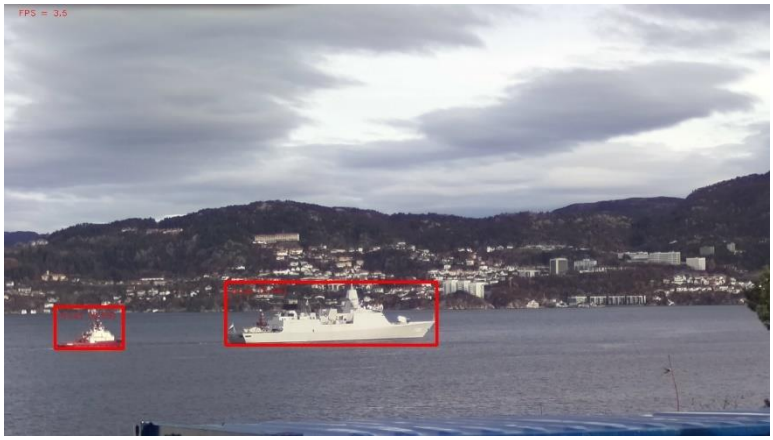
	MODELL			
	efficientDet_lite 0	efficientDet_lite 1	efficientDet_lite 2	ssd_mobilenet_v2 FPN-lite
	FPS			
	5,3	3,5	2,4	2
DETEKSJONER	GJENNOMSNITLIG SANNSYNLIGHET			
1	0,33	0,36	0,6	0,8
2	0,5	0,38	0,58	0,87
3	0,5	0,36	0,58	0,56
4	0,54	0,36	0,66	0,62
5	0,52	0,61	0,68	0,8
6	0,52	0,61	0,68	0,79
7	0,46	0,62	0,64	0,99
8	0,5	0,62	0,66	0,84
9	0,54	0,61	0,68	0,98
10	0,46	0,56	0,6	0,88
11	0,46	0,61	0,54	0,7
12	0,35	0,59	0,54	0,69
13	0,37	0,5	0,6	0,85
14	0,35	0,61	0,46	0,73
15	0,37	0,61	0,5	0,96
16	0,33	0,59	0,38	0,92
Gjennomsnitt i %	44,375	53,75	58,625	81,125

Vedlegg I – Testbilder fra efficientDet_lite 0





Vedlegg J - Testbilder fra efficientDet_lite 1



Vedlegg K - Testbilder fra efficientDet_lite 2

Vedlegg L - Testbilder ssd_mobilenet_v2 FPN-lite