Bacheloroppgave

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

Birger Johan Nordølum
Eirik Osland Lavik
Kristian André Dahl Haugen
Tom-Ruben Traavik Kvalvaag

Artsgjenkjenning av fisk

Bacheloroppgave i Bachelor i ingeniørfag - Data
Veileder: Marius Pedersen

Mai 2021

NTNU
Kunnskap for en bedre verden

Birger Johan Nordølum
Eirik Osland Lavik
Kristian André Dahl Haugen
Tom-Ruben Traavik Kvalvaag

Artsgjenkjenning av fisk

Bacheloroppgave i Bachelor i ingeniørfag - Data
Veileder: Marius Pedersen
Mai 2021

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

NTNU
Kunnskap for en bedre verden

Abstract

Norwegian Institute of Nature Research (NINA) inquired about a tool to automatically detect underwater species in videos, and calculate basic statistics of their activity. Something that would assist them in their research. This project investigates possibilities and implements a solution. A dataset of the relevant species is made for use in the development and testing of object tracking and detection algorithms. The solution implements an object detection module where deep learning is used to detect an object by using a You Only Look Once(YOLO) based network architecture. The detections are used in a tracking module where Simple Online and Realtime Tracking(SORT) is used to associate detections over time to generate individual objects observed in the video. In order to organize the object detection and tracking were a core module implemented. This module processes and then stores the results in a database. A web-based user interface communicates to the core module to administer, view, and download results. The proposed solution gives NINA a working tool for use in their research.

Sammendrag

Norsk institutt for naturforskning (NINA) ønsket en løsning som kunne automatisk gjenkjenne arter i undervannsvideo og gi de grunnleggende statistikk om aktiviten i videoene. Dette prosjektet undersøker muligheter for å løse problemet og implementerer en løsning. Et dataset ble annotert med artene NINA var interessert i for bruk i utvikling og test av deteksjon- og sporingalgoritmer. Løsningen implementerer en objektdeteksjon modul hvor dyplæring brukes for å detektere objekter ved bruk av et nettverk basert på You Only Look Once (YOLO)-nettverksarkitektur. Deteksjonene brukes i en sporing modul hvor Simple Online and Realtime Tracking (SORT) er brukt for å assosiere deteksjoner over tid for å generere individuelle objekt observert i videoen. Organisering av objektdeteksjon og sporing er implementert i en kjernemodul som organiserer prosessering og lagrer resultat til en database. Et webbasert brukergrensesnitt kommuniserer med kjernemodulen som lar bruker administrere, se og laste ned resultat av prosesserte videoer. Den totale løsningen gir NINA et fungerende verktøy for bruk i sin forskning.

Forord

Oppgaven er skrevet av Birger Johan Nordølum, Eirik Osland Lavik, Kristian André Dahl Haugen og Tom-Ruben Traavik Kvalvaag ved institutt for datateknologi og informatikk ved NTNU i Gjøvik.

Vi ønsker å takke de som har støttet oss gjennom oppgaven. Først ønsker vi å utrette en takk til vår veileder, Marius Pedersen ved NTNU i Gjøvik. Gjennom ukentlige møter har Marius vært til stor hjelp. Gruppen ønsker også å takke vår oppdragsgiver, Norsk institutt for naturforskning (NINA), representert ved Knut Marius Myrvold og Tobias Holter. De ga oss muligheten og motivasjon til å ta fatt på denne spennende oppgaven. Gjennom hele prosessen har oppdragsgiver vært støttende og vist stor interesse i arbeidet vi har gjort. Vi vil også rette en takk til medstudenter, familie og venner. Deres hjelp til å lese gjennom rapporten for kvalitetssikringen har vært uerstattelig.

Til slutt og ikke minst ønsker vi å takke oss selv for et godt samarbeid gjennom disse fem månedene. Det har vært en spennende og utfyllende prosess.

Innhold

Figurer

Tabeller

Kodelister

Akronymer

MOTA Multiple Object Tracking Accuracy. 3, 4, 27, 53, 54, 101

MOTP Multiple Object Tracking Precision. 3, 4, 27, 53, 54, 101

mAP mean Average Precision. 3, 4, 25, 45, 101

API application programming interface. xiii, 18, 40, 41, 44, 47, 55, 100

CLI Command-line interface. xviii, 27, 187

COCO Common Objects in Context. 36, 37, 39

CSS Cascading Style Sheets. 4, 27, 28, 30, 69, 74, 77, 78

CVAT Computer Vision Annotation Tool. 37, 43, 44

Datumaro Dataset Management Framework. 37, 43

DDD Domain Driven Design. 57, 59

EER Enhanced Entity-Relationship. 31

GPU Graphics Processing Unit. 39, 41, 47, 99, 100

HTML HyperText Markup Language. 4, 27, 28, 30, 69, 74, 76

HTTP Hypertext Transfer Protocol. 33, 73, 99, 100

IMRAD Introduction, Methods, Results, and Discussion. 7, 35

JSON JavaScript Object Notation. 59, 77

NINA Norsk institutt for naturforskning. v, vii, 1–3, 17, 61, 62, 101–103, 189

ORM Object Relational Mapper. 31, 70

SORT Simple Online and Realtime Tracking. v, xvii, 26, 51, 53–55, 100, 164

VM Virtual Machine. xv, 39, 41, 47

YOLO You Only Look Once. v, 21, 23, 24, 35, 36, 100

Ordliste

avgrensningsboks (bounding box) En tenkt avgrensningsboks rundt et objekt av interesse. 23, 24, 26, 27, 35, 37, 49, 51, 54, 55

brukermønster (use case) Innen systemutvikling beskriver hvordan eksterne brukere og systemer samhandle med løsningen.. 9

hyperparameter 'hyperparameter" er innstillinger som kontrollerer en maskinlæringsalgoritme sin oppførsel [1, s. 118]. 39

Intersection over Union Overlapp mellom antatt avgrensningsboks og faktisk avgrensningsboks. xiii, 26

JavaScript JavaScript er et programmersspråk som kan kjøre i nettleser og endre DOM. 28

konvolusjonalt nevrale nettverk Nevral nettverk som bruker konvolusjon operasjoner mellom lagene i nettverket. 35

Poetry En pakkebehandler for Python. 95

Static program analyser Verktøy for å analysere kildekode uten å kjøre det. 87

Kapittel 1

Introduksjon

1.1 Bakgrunn

Norsk institutt for naturforskning (NINA)[1] benytter videoovervåking for å kunne passivt og skånsomt undersøke natur på land og i vann. Tidligere har NINA overvåket fisketrapper [2], og fått utviklet teknologier for å analysere innsamlet videomateriell automatisk [3]. Denne løsningen kunne detektere om fisken i fisketrapp var villfisk eller settefisk.

I 2020 utvidet NINA overvåkning til å inkludere komplekse fiskesamfunn i sesongmessige habitat. Med dette har NINA samlet inn rundt 1290 timer med videomateriell som må undersøkes. Forsøk på å analysere dette manuelt er en svært tidkrevende jobb for forskerne som må se igjennom videomateriellet i nær sanntid. Se regneark i figur 1.1 for eksempel på manuell gjennomgang av video ved tidligere prosjekt hos NINA.

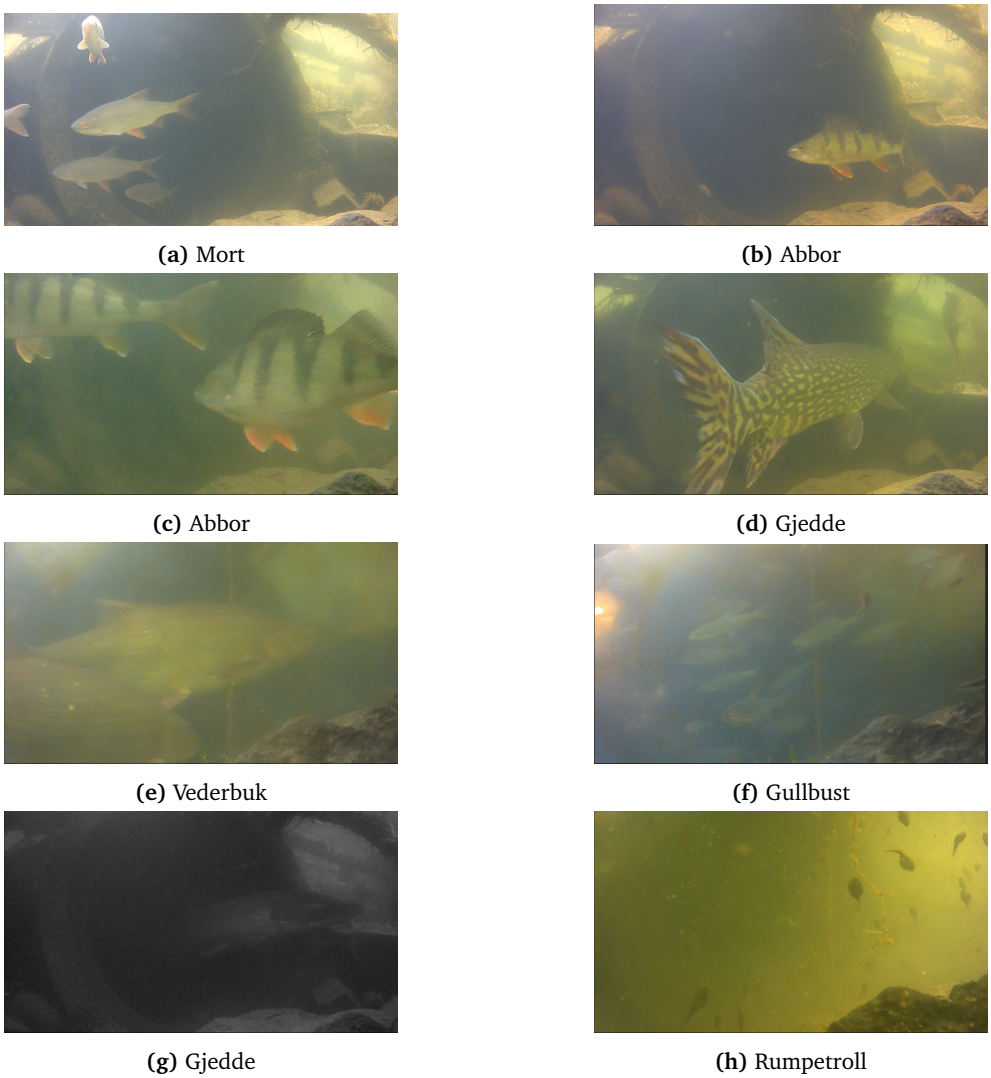
	A	B	C	D	E	F	G	H	I	J	K	
1	Observasjon	ID	Date	Time	Sortname	Direction	Disk	Filnavn	Første gang	Andre tidspunkt av samme individ	Sikt	Kommetar
2	1	Harr170505_1	05/05/2017	14:34 - 15:04	Harr	Opp	1	File1-[2017-05-04_13-34-27]-000	16:10		Dårlig	Kom inn i kar
3	2	Harr170505_2	05/05/2017	15:04 - 15:34	Harr	Kam	1	File1-[2017-05-04_13-34-27]-001	01:44	01:49; 02:12; 02:20	Dårlig	Ikke mulig å s
4	3	Fisk170505_1	05/05/2017	15:04 - 15:34	Fish	Kam	1	File1-[2017-05-04_13-34-27]-001	07:04	08:37; 08:54	Dårlig	Ikke mulig å s
5	4	Harr170505_3	05/05/2017	15:04 - 15:34	Harr	Kam	1	File1-[2017-05-04_13-34-27]-001	09:00	09:24; 09;49; 10:01; 11:01; 11:26, 11:51	Dårlig	to harr glir in
6	5	Harr170505_4	05/05/2017	15:04 - 15:34	Harr	Kam	1	File1-[2017-05-04_13-34-27]-001	09:00	09:24; 09;49; 10:01; 11:01; 11:26, 11:51	Dårlig	to harr glir in
7	6	Fisk170505_2	05/05/2017	15:04 - 15:34	Fish	Kam	1	File1-[2017-05-04_13-34-27]-001	17:47		Dårlig	Liten fisk mer
8	7	Harr170505_5	05/05/2017	15:04 - 15:34	Harr	Opp	1	File1-[2017-05-04_13-34-27]-001	23:23		Dårlig	Kommer inn
9	8	Harr170505_5	05/05/2017	15:04 - 15:34	Harr	Kam	1	File1-[2017-05-04_13-34-27]-001	25:09		Dårlig	Glir inn fra hy
10	9	Harr170505_6	05/05/2017	15:34- 16:04	Harr	Opp	1	File1-[2017-05-04_13-34-27]-002	00:39		Dårlig	oppvandrenc
11	10	Harr170505_7	05/05/2017	15:34- 16:04	Harr	Kam	1	File1-[2017-05-04_13-34-27]-002	15:26		Dårlig	Vanskelig og
12	11	Harr170505_8	05/05/2017	15:34- 16:04	Harr	Kam	1	File1-[2017-05-04_13-34-27]-002	18:25	18:39	Dårlig	Glir inn fra hy
13	12	Harr170505_9	05/05/2017	16:04 - 16:34	Harr	Opp	1	File1-[2017-05-04_13-34-27]-003	00:48		Dårlig	oppvandrenc
14	13	Harr170505_10	05/05/2017	16:04 - 16:34	Harr	Kam	1	File1-[2017-05-04_13-34-27]-003	02:27	02:39; 02:55	Dårlig	Ikke mulig å s
15	14	Harr170505_11	05/05/2017	16:04 - 16:34	Harr	Opp	1	File1-[2017-05-04_13-34-27]-003	09:12		Dårlig	oppvandrenc
16	15	Harr170508_1	08/05/2017	13:05 - 13:35	Harr	Kam	1	File1-[2017-05-04_13-34-27]-141	01:15	01:51; 02:03	Dårlig	kommer inn t
17	16	Harr170508_2	08/05/2017	13:05 - 13:35	Harr	Kam	1	File1-[2017-05-04_13-34-27]-141	03:59		Dårlig	kommer inn t
18	17	Harr170508_2	08/05/2017	13:05 - 13:35	Harr	Kam	1	File1-[2017-05-04_13-34-27]-141	05:47	06:45	Dårlig	muligens tidl
19	18	Harr170508_3	08/05/2017	13:05 - 13:35	Harr	Ned	1	File1-[2017-05-04_13-34-27]-141	08:14		Dårlig	Nedstrøms
20	19	Harr170508_4	08/05/2017	13:05 - 13:35	Harr	Kam	1	File1-[2017-05-04_13-34-27]-141	09:39	10:10; 11:11;	Dårlig	kommer inn t
21	20	Harr170508_4	08/05/2017	13:05 - 13:35	Harr	Kam	1	File1-[2017-05-04_13-34-27]-141	13:29	13:53	Dårlig	kommer inn t
22	21	Harr170508_4	08/05/2017	13:05 - 13:35	Harr	Kam	1	File1-[2017-05-04_13-34-27]-141	18:15	18:34; 19:29, 20:34, 23:09	Dårlig	kommer inn t
23	22	Harr170508_5	08/05/2017	13:05 - 13:35	Harr	Opp	1	File1-[2017-05-04_13-34-27]-141	23:42		Dårlig	oppvandrenc
24	23	Harr170508_5	08/05/2017	13:05 - 13:35	Harr	Kam	1	File1-[2017-05-04_13-34-27]-141	24:44	25:31; 26:01; 28:51	Dårlig	kommer inn t
25	26	Harr170818_1	18/08/2017	12:47 - 13:17	Harr	Opp	1	File1-[2017-08-18_11-46-59]-000	04:09		God	Litt vanskelig
26	27	Harr170818_2	18/08/2017	12:47 - 13:17	Harr	Opp	1	File1-[2017-08-18_11-46-59]-000	04:49	05:10	God	Fin passering
27	28	Harr170818_1	18/08/2017	12:47 - 13:17	Harr	Kam	1	File1-[2017-08-18_11-46-59]-000	05:21		God	To harr nede
28	29	Harr170818_2	18/08/2017	12:47 - 13:17	Harr	Kam	1	File1-[2017-08-18_11-46-59]-000	05:21		God	To harr nede

Figur 1.1: Regneark fra tidligere gjennomgang av videomateriell.

[1] `https://nina.no`, besøkt 29.01.2021

I videomateriellet samlet inn i 2020 er det i hovedsak ni arter som NINA har interesse for, som kan sees i figur 1.2 og liste under.

- Abbor
- Brasme
- Gjedde
- Gullbust
- Mort
- Rumpetroll
- Stingsild
- Vederbuk
- Ørekyt



(a) Mort **(b)** Abbor **(c)** Abbor **(d)** Gjedde **(e)** Vederbuk **(f)** Gullbust **(g)** Gjedde **(h)** Rumpetroll

Figur 1.2: Eksempler på arter og forhold i videomateriellet.

1.2 Prosjektbeskrivelse fra oppdragsgiver

Med bakgrunn i videomateriellet fra sesongmessige habitat ønsker oppdragsgiver et verktøy som automatisk kan gjenkjenne arter. Dette inkluderer video fra forskjellige vann og lysforhold. Løsningen skal også kunne loggføre antall arter gjenkjent innenfor et tidsrom. Oppdragsgiver ønsker også et brukergrensesnitt til verktøyet. For den fulle oppgavebeskrivelsen fra oppdragsgiver og bakgrunnsinformasjon se Vedlegg B.

1.3 Mål og rammer

For å løse bacheloroppgaven er det nødvendig å definere tydelige mål og rammer for oppgaven. Gjennom denne seksjonen vil gruppen formidle hvilke begrensinger og mål som er viktig for å løse oppdragsgivers problemstilling. Mye av disse var definert fra prosjektplanen, se vedlegg C.

1.3.1 Rammer

Tiden gruppen har tilgjengelig er en av de mest kritiske ressursene. Ut fra prosjektplanen er tidsrammen til bacheloroppgaven mellom 11. Januar 2021 og 20. Mai 2021. Løsningen skal utvikles fra oppgavestart, til utviklingstopp 23. April 2021.

Basert på dialog med NINA, var følgende teknologirammer identifisert.

- Løsningen må fungere på Windows
- Løsningen skal ha et enkelt og intuitivt brukergrensesnitt
- Løsningen skal fungere på en selvstendig arbeidsstasjon
- Løsningen skal detektere følgende arter: Abbor, Brasme, Gjedde, Gullbust, Mort, Rumpetroll, Stingsild, Vederbuk og Ørekyt

1.3.2 Resultatmål

Det skal produseres en løsning for å detektere, spore og artsbestemme fisk i en videostrøm. Gjennom et enkelt og intuitivt brukergrensesnitt, skal brukeren kunne administrere løsningen. Løsningen skal returnere en rapport som viser antall fisk av bestemt art, tidspunkt den kom inn i, og når den forlot bilde. Denne rapporten skal være på et format som kan enkelt importeres inn i andre program for videre analyse.

- **Deteksjon/artsbestemming** trenger høy presisjon. Dette er for å sikre at dersom en art vises i videoen, skal dens posisjon detekteres og klassifiseres. Målet er en mean Average Precision (mAP), $mAP_{IoU=0.5} = 0.6$.
- **Sporing** anses å være like viktig som deteksjon. Ved dårlig sporing kan det samme objektet telles flere ganger. Målet er for Multiple Object Tracking Accuracy ($MOTA$) blir 60. Multiple Object Tracking Precision ($MOTP$) settes til 75.

- **Menneskelig tid** ønskes å være svært lav. I dag ligger den på ett minutt per minutt. Basert på øvrige mål skal dette reduseres til 5 minutt per 30 minutt video.

Mål for *mAP* er basert på resultat fra en tidligere oppgave med tilsvarende problemstilling [4]. For $MOTA$ og $MOTP$ er det basert på resultat fra tester utført på Multiple Object Tracking Benchmark[2].

1.3.3 Effektmål

Det beste utfallet av bacheloroppgaven, er at det resulterer i en løsning som betydelig reduserer tiden oppdragsgiver bruker på å analysere videofiler. Et annet mål er at løsning skal være utvidbar. Dette legger til rette for at senere bacheloroppgaver kan utvide løsningen med ny funksjonalitet. Det er også viktig siden løsningen skal publiseres.

Derfor setter gruppen følgene som effektmål på bacheloroppgaven:

- Frigjøre arbeidskapasitet hos oppdragsgiver.
- Kunne videreutvikles av andre enn de som var involvert i prosjektet.
- Løsningen skal kunne desentraliseres slik at den kan kjøre på flere maskiner eller lokasjoner.

1.3.4 Avgrensning

Bruker- og tjenerautentisering blir ikke del av prosjektet. Brukergrensesnittet blir gjort tilgjengelig på arbeidstasjonen hvor løsningen kjører, eller lokalt på nettverket om det kjører på en server. Dersom uvedkommende får tilgang er det ikke annet å se enn data fra tidligere jobber eller starte og stoppe prosesseringsjobber.

Prosjektet avgrenses til å artsgjenkjenne artene som definert i avsnitt 1.1, hvor overlevert videomaterialet er tilstrekkelig.

1.4 Gruppens bakgrunn

Gruppen består av fire studenter ved studieprogrammet Bachelor i ingeniørfag - data, ved NTNU i Gjøvik. Alle medlemmene har samme grunnlag fra obligatoriske emner i studieplanen. Det som skiller er valg av valgfrie emner og tidligere erfaring. Ett av medlemmene har tatt emner innen matematikk, datasyn og kunstig intelligens. De tre resterende innen utvikling og programmerbar infrastruktur. Fra tidligere har ett av medlemmene erfaring med bruk av HyperText Markup Language (HTML) og Cascading Style Sheets (CSS).

[2] `https://motchallenge.net/`, besøkt 20.01.21

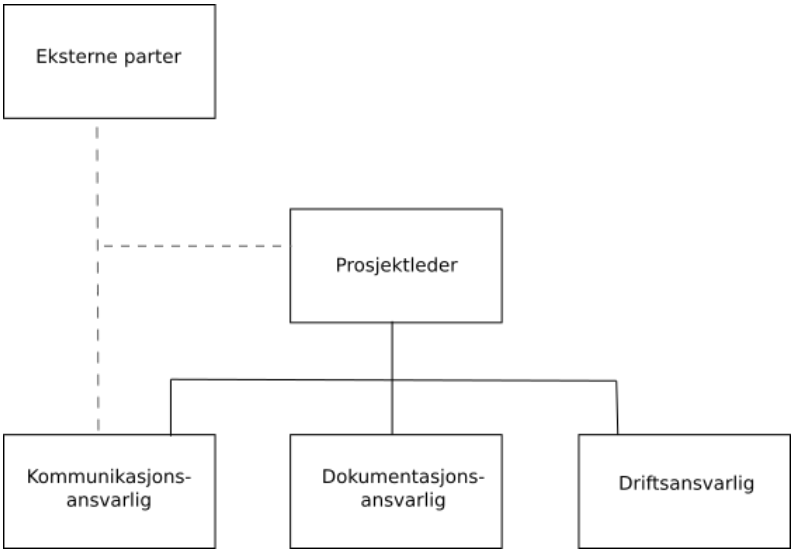
1.5 Organisering

Som del av prosjektplanleggingen ble organiseringen av prosjektet, interne roller og ansvarsforhold avtalt. Relevant informasjon fra prosjektplan er gjengitt her og den fulle prosjektplanen kan sees i Vedlegg C.

På grunn av smittesituasjonen i forbindelse med *COVID-19* var det planlagt å gjennomføre prosjektet digitalt. Eventuelt på campus etter avtale internt i gruppen, om dette var tillatt.

1.5.1 Prosjektroller

Internt i prosjektet var det definert noen prosjektroller med tilhørende ansvarsområder. Dette var gjort for å sikre at viktige elementer i prosjektarbeidet blir fulgt opp. Disse er definert i listen under og Figur 1.3.



Figur 1.3: Organisasjonskart for internt i gruppe

- **Prosjektleder** - Tom-Ruben T. Kvalvaag
  - Holde overordnet oversikt til prosjektet og styre arbeid i riktig retning som avtalt.
  - Siste ord ved avstemming uten flertall.
  - Ordstyrer i møter.
- **Dokumentansvarlig** - Kristian A. D. Haugen
  - Passe på at møter blir dokumentert og fulgt opp.
  - Oversikt av dokumentasjon i kode og organisere utvikling av brukerveiledning.
- **Kommunikasjonsansvarlig** - Birger J. Nordølum

    - Ansvarlig for kommunikasjonen ut fra og inn til gruppa.
- **Driftsansvarlig** - Eirik O. Lavik
    - Ansvar for å drifte infrastrukturen rundt miljøet som brukes i prosjektet.

I tillegg til internroller, var hvert medlem å betrakte som en utvikler.

1.5.2 Faglige ansvarsområder

Som en del av kvalitetssikringen hadde hvert gruppemedlem sitt faglige ansvarsområde de fordypet seg innen. Den ansvarlige undersøkte hva som var kunnskapsfront av forskning og teknologier innen fagfeltet.

Ut fra de områdene som var vurdert som kritiske for løsningen ble følgende fagområder valgt

- **Brukergrensesnitt og brukeropplevelse** - Birger J. Nordølum
- **Maskinlæring**
    - **Deteksjon** - Tom-Ruben T. Kvalvaag
    - **Sporing** - Eirik O. Lavik
- **Batchprosessering** - Kristian A. D. Haugen

1.6 Rapportens oppbygging

Rapporten er bygget opp for å leses sekvensielt fra kapittel til kapittel. Her beskrives en kort oversikt over innholdet i hvert kapittel. Det gis også eventuelle anbefalinger for kapitler som burde leses først.

Kapittel 1 - Introduksjon

Introduserer prosjektet, oppgaven og gir bakgrunn for problemstillingen. Presenterer gruppen sin organisering og bakgrunn.

Kapittel 2 - Kravspesifikasjon

Kapittelet definerer kravene til løsningen ut fra mål og rammer. Det blir også gjennomført en kartlegging av systemets hovedfunksjonalitet.

Kapittel 3 - Utviklingsprosess

Gjennomgår plan for utviklingsprosess og hvordan den har blitt gjennomført. Her nevnes det også hvordan gruppen har utført møter med eksterne parter.

Kapittel 4 - Teori og teknologier

Gjennomgår teori og teknologier brukt som grunnlag i prosjektet. Hvor tema som er presentert gir litt forkunnskap for senere kapittel.

Kapittel 5 - Design og implementasjon

Gir en overordnet oversikt over løsningen og dens komponenter. Bør leses før kapittel 6-9 for å få en forståelse av sammenhengen mellom disse.

Kapittel 6 - Objektdeteksjon

Kapittelet beskriver hvordan løsningen implementerer deteksjon av objekter. Bruker teori og konsepter presentert i avsnittene 4.1.1 og 4.1.2.

Kapittel 7 - Sporing

Her skrives om implementeringen av objektsporing. Bruker teori og konsepter presentert i avsnittene 4.1.4 og 4.1.5.

Kapittel 8 - Kjernefunksjonalitet

Her forklares hvordan handteringen av analysert data gjennomføres, og hvordan video blir prosessert. Bruker teknologier presentert i avsnitt 4.2.8. Dette er løsningens største modul og inneholder hvordan hovedfunksjonalitet i programmet er implementert.

Kapittel 9 - Brukergrensesnitt

Kapittelet er bygd som en egen del av rapporten med sin egen IMRAD-struktur. Bruker teori og konsepter presentert i brukergrensesnitt avsnitt 4.1.6, Flask avsnitt 4.2.5 og Tailwind CSS avsnitt 4.2.6.

Kapittel 10 - Kvalitetssikring

Gjennomgang av de ulike verktøyene som blir brukt for å ivareta kvalitet på koden og hvordan gruppen har jobbet sammen opp mot kodebrønnen.

Kapittel 11 - Diskusjon

Diskuterer løsningen og prosjektet som en helhet.

Kapittel 12 - Konklusjon

Konkluderer om løsning og prosjektet har oppnådd satte mål.

Kapittel 2

Kravspesifikasjon

Ut fra oppgaveskrivelsen, mål og rammer definert i kapittel 1 er det laget krav til løsningen. I dette kapittelet vil disse spesifikasjonene bli dokumentert.

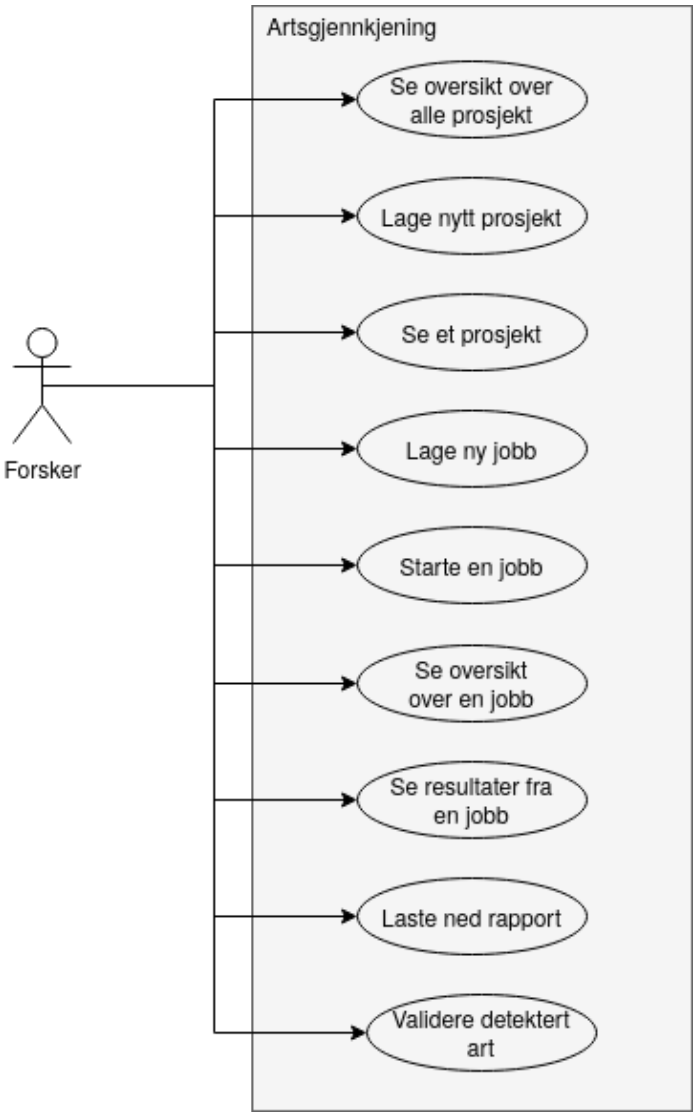
2.1 Funksjonelle krav

2.1.1 Brukermønster

Ut fra oppgaveskrivelsen som definert av oppdragsgiver i avsnitt 1.2 ble et sett av brukermønster utviklet og presentert til oppdragsgiver. Se høynivå brukermønster i tabeller 2.1 til 2.9 og diagram i figur 2.1. Disse representerer oppgavene forskerne hos oppdragsgiver har forespurt i oppgaveskrivelsen og gjennom samtaler i møte.

De definerte brukermønstre i tabeller 2.1 til 2.9 ble brukt som utgangspunkt i utviklingen når ny funksjonalitet ble valgt i en *iterasjon* for implementering. Hvor de var grunnlaget for å generere arbeidskort til iterasjonene, som diskutert i kapittel 3.

Ut fra brukermønstrene ble det også definert en minimumsløsning som var brukermønster i tabeller 2.1 til 2.8.



Figur 2.1: Brukermønster(use case) diagram

Tabell 2.1: Brukermønster nummer 1: oversikt over prosjekt.

Brukermønster:	Se oversikt over alle prosjekt.
Aktør:	Forsker
Mål:	Få oversikt over alle prosjekt.
Beskrivelse:	Viser oversikt over alle prosjekt.

Tabell 2.2: Brukermønster nummer 2: lage nytt prosjekt

Brukermønster:	Lage nytt prosjekt
Aktør:	Forsker
Mål:	Lage et nytt prosjekt
Beskrivelse:	Lage et nytt prosjekt med navn, prosjekt-nummer og beskrivelse.

Tabell 2.3: Brukermønster nummer 3: lage ny jobb

Brukermønster:	Lage ny jobb
Aktør:	Forsker
Mål	Planlegge en ny jobb for prosessering i et prosjekt.
Beskrivelse:	Legger inn en ny jobb med navn, lokasjon, beskrivelse og videoer.

Tabell 2.4: Brukermønster nummer 4: laste ned rapport

Brukermønster:	Last ned rapport
Aktør:	Forsker
Mål	Sjå resultat fra analyse
Beskrivelse:	På side for valgte jobb, laste ned *csv* fil med resultater.

Tabell 2.5: Brukermønster nummer 5: starte en jobb

Brukermønster:	Starter en jobb
Aktør:	Forsker
Mål	Legger jobb i kø for prosessering.
Beskrivelse:	På side for valgte jobb, starter eller legger jobb i kø for prosessering.

Tabell 2.6: Brukermønster nummer 6: se et prosjekt

Brukermønster:	Se et prosjekt.
Aktør:	Forsker
Mål	Ser oversikt for valgt prosjekt.
Beskrivelse:	For valgt prosjekt, ser oversikt over navn, prosjekt-nummer, beskrivelse og assosierte jobber.

Tabell 2.7: Brukermønster nummer 7: se en jobb

Brukermønster:	Se oversikt over en job.
Aktør:	Forsker
Mål	Ser oversikt for valgte jobb.
Beskrivelse:	For valgte jobb, ser oversikt over navn, lokasjon, beskrivelse, videoer, status og progresjon.

Tabell 2.8: Brukermønster nummer 8: se resultater fra jobb

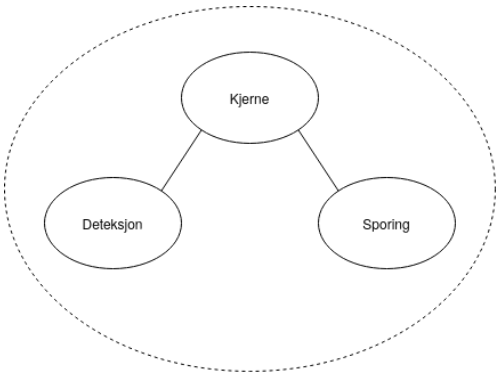
Brukermønster:	Se resultater fra en jobb.
Aktør:	Forsker
Mål	Inspisere resultat fra en jobb.
Beskrivelse:	På side for valgte job, ser tabell over resultat av ferdig prosessert jobb.

Tabell 2.9: Brukermønster nummer 9: validere deteksjoner

Brukermønster:	Validere detektert art.
Aktør:	Forsker
Mål	Validere detektert art.
Beskrivelse:	På side for valgte job, i tabell over resultat, vise bilde eller video av detektert art for kvalitetsikring.

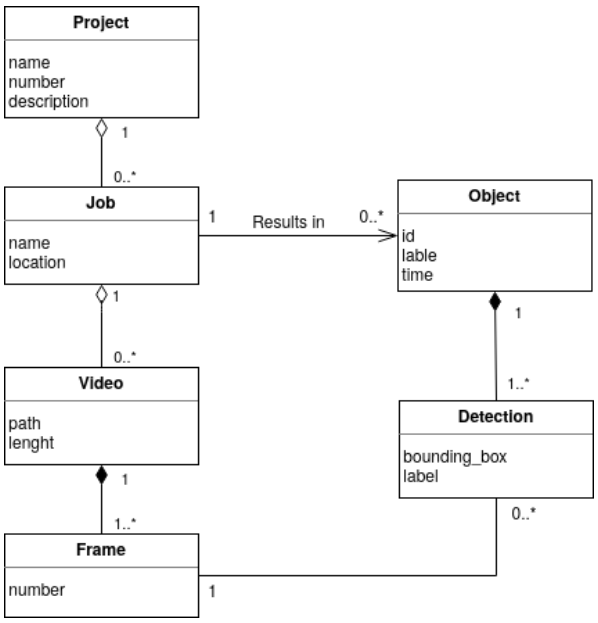
2.1.2 Domene modell

Under planlegging av prosjektet ble oppgavene definert ut fra brukermønstrene. På grunnlag av disse, ble oppgaven delt inn i tre domener. De tre domenene er illustrert i figur 2.2. Basert på dette ble det gjort en system modellering av løsningen.



Figur 2.2: Domene oversikt i løsningen

Av de tre domenene er det kjernefunksjonaliteten som inneholder forretningslogikken. Her blir ønskede operasjoner organisert og resultater lagret, se domenemodell i figur 2.3. Kjernen bruker de andre domenene for å oppnå ønsket resultat. Disse er små domener som er spesialisert på en oppgave, se diagram i figur 2.4.

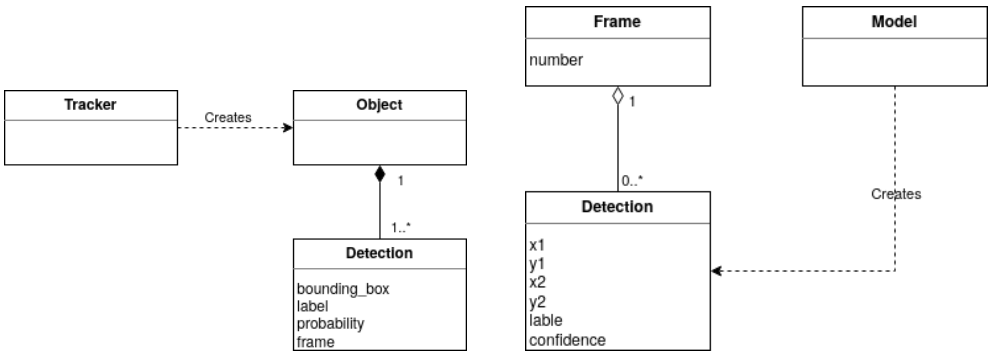


Figur 2.3: Domenemodell av kjernefunksjonalitet

2.2 Ikke-funksjonelle krav

Det er definert noen ikke-funksjonelle krav til løsningen. Den skal

- kunne kjøres på en arbeidsstasjon med operativsystemet Microsoft Windows 10.



(a) Sporing **(b)** Deteksjon

Figur 2.4: Domenemodell av deteksjon og sporing

- designes for å kunne kjøre deler av den distribuert ved videreutvikling i fremtiden. Dette prosjektet har ikke som mål at leveransen skal kunne kjøres distribuert, men at arkitektur og design legger til rette for det.
- ha et enkelt og intuitivt brukergrensesnitt.

2.3 Operasjonelle krav

2.3.1 Åpen kildekode

Gruppen er veldig opptatt av åpen kildekode. Det ble foreslått til oppdragsgiver om å utvikle løsning som åpen kildekode. Dette var noe de stilte seg bak. Dette valget medførte at noen krav måtte spesifiseres. «Study of the Release Process of Open Source Software: Case Study» [5] har vært med som inspirasjon for å forfatte kravene.

Ved at kildekoden skal være åpen, er det formalisert noen krav som gruppen mener er sentrale når et prosjekt skal publiseres. Det er to viktige momenter her. Hvordan koden skrives og hvordan den integrerer andres kode. Ved bruk av andres kode er det veldig viktig at det ikke brukes lisenserte løsninger som ikke er forenelig med den ønskede lisensen. Les mer om valget av lisens i kapittel 13. Fra dette er kravene definert til at kodebasen skal

- være ryddig og bestå av en konsis stil
- være så godt dokumentert som mulig
- følge de standarder som enten er offisielle eller tatt i bruk av store prosjekter
- være slik at det skal være enkelt for andre å komme senere for å bidra til koden
- være kodet og dokumentert på engelsk for å gjøre prosjektet mer attraktivt

Kapittel 3

Utviklingsprosess

I prosjektplanen, vedlegg C, går gruppen gjennom hvordan utførelsen av prosjektet skal være. Dette kapitelet bygger videre på planen og oppsummerer gjennomføringen.

3.1 Møter

Gjennom prosjektet har gruppen hatt ulike type møter. Disse tre var møte mellom gruppen internt, møte sammen med veileder, og sammen med oppdragsgiver. Dette sikret at alle fikk vite og kjente til status av utviklingen gjennom av oppgaven. Alle møtene ble arrangert digitalt via *Zoom*. Se vedlegg P for eksempler av møtereferater.

3.1.1 Morgenmøte

Hver dag fra tirsdag til fredag kl. 09:30 hadde gruppen morgenmøte. Under møtet la alle frem hva som var blitt gjort dagen før og hva som var planen for denne dagen. Møtene ble planlagt til å maksimalt vare i 20 minutter. Det var gruppeleders ansvar å heve møtet etter alle var ferdig. Etter møtet var det mulig å starte opp en diskusjon, eller spørre om hjelp hvis noe var uklart eller sto fast på. Møtene fungerte veldig bra til å få oversikt over hva hvert medlem jobbet på.

3.1.2 Veiledningsmøte

Hver tirsdag kl. 09:00 var det møte med gruppens veileder, Marius Pedersen. Her informerte gruppen om progresjon og stilte spørsmål som hadde kommet den siste uken. Ofte ble det er spurt om datasyn og maskinlæring

3.1.3 Iterasjonsmøte

Etter hver iterasjon, se neste delkapittel avsnitt 3.2.1, ble det holdt møte med oppdragsgiver. Under møtet ble det gjennomgått hva som var blitt gjort den siste

iterasjonen. På møtene gikk gruppen ofte gjennom demoer av ny funksjonalitet. Det ble gitt mulighet til å kommentere på arbeidet. I tillegg var det vanlige diskusjon. Disse møtene var veldig nyttige da det utviklingen mellom møtene foregikk hyppig . Mellom møtene hadde vi en liten agende for hva som skulle gjøres til neste møte.

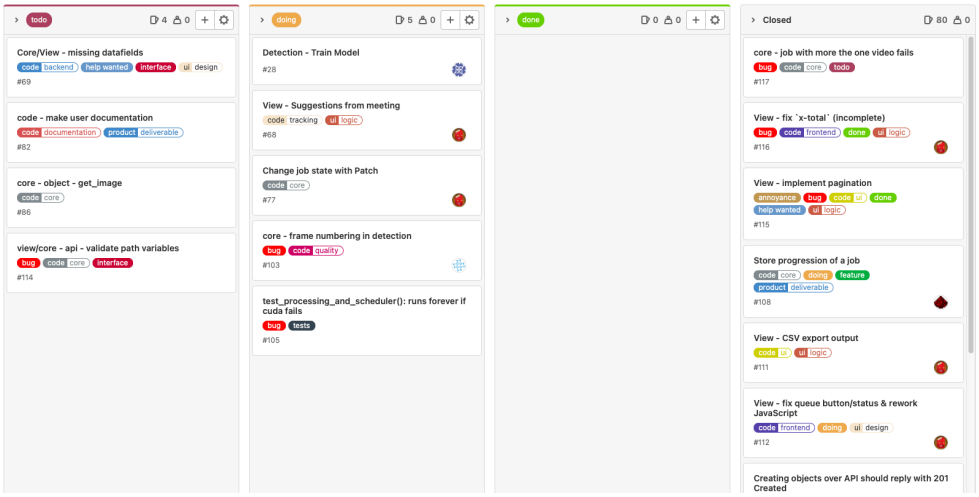
Gruppen og arbeidsgiver hadde et møte den 16. desember hvor formålet var å bli kjent og smått begynne å diskutere ideer. I forkant hadde gruppen diskutert seg i mellom hva og hvordan løsningen kunne bli.

3.2 Utviklingsmodell

Scrumban ble brukt som utviklingsmodell, se vedlegg C for en utdypende argumentasjon for dette. Valgt utviklingsmodell kombinerer noen konsepter fra Kanban og Scrum. Gruppen har brukt Kanban-brett sammen med iterasjoner.

Et kanban-brett er et organiseringsverktøy for utvikling. Verktøyet kan brukes til å kategorisere og prioritere arbeidsoppgaver. Hver arbeidsoppgave er representert med ett kort. Gjennom utviklingen ble systemet brukt til å distribuere og holde oversikt over arbeid. Ved hjelp av dette vil gruppen alltid ha oversikt over hvilke oppgaver som må gjøres, samt hvem som jobber med de.

Oppgavene ble beskrevet ved hjelp av et kort. Et kort er representert som et *issue* på GitLab. Disse innholder en overordnet beskrivelse over hva oppgaven går ut på. Kortet beskriver oppgaven ved å først gi en kort oppsummering og deretter en liten liste over punkter som inngår i oppgaven. I tillegg er det mulig å kommentere på kortet slik at diskusjon er på ett sted og er kun relevant til oppgaven. Bruk av kort har vært med å gjøre det enkelt å holde alle relevant informasjon på ett sted.



Figur 3.1: Gruppens Kanban-brett.

Stikkord ble brukt til å organisere kortene. Disse kan tildeles kortene på kanban-

brettet. Ved å bruke stikkord kan man enkelt se statusen på et kort. Dette viktig for gruppen når et medlem var ferdig med sin oppgave og var klar til å jobbe på en ny oppgave. Med et stikkord som fortalte status, hindret dette at flere begynte på samme oppgave. Det gav også medlem informasjon når funksjonalitet var ferdig og hadde behov for en gjennomgang.

Figur 3.1 viser gruppens Kanban-brett. Status-stikkordet er satt til å være representert som kolonner. Fra venstre til høyre er *todo*, *doing*, *done* og *closed*. I tillegg ser man alle stikkordene hvert kort har blitt tildelt. Disse er representeres som runde bokser under tittelen. Unike farger benyttes for å lett gjenkjenne stikkordene. Det fine med stikkord er at relevant informasjon er fortalt med få ord.

3.2.1 Gjennomføring av iterasjon

Iterasjonene deler opp prosjektet i klare milepæler. En iterasjon varer i to uker, og ved slutten blir det gjennomført et møte med oppdragsgiver for å vise progresjon.

I starten av en iterasjon holdes et iterasjonsmøte. Her gjennomgås først kortene fra forrige iterasjon. Dersom kortene er ferdig flyttes de til *closed*. Om de trenger mer arbeid flyttes de enten til *todo*, eller blir værende og gjelder for neste iterasjon. Neste iterasjon planlegges ved å prioritere kort fra *todo*. Det ble ikke gjennomført en fast prosedyre, men hvert medlem fortale om hva de mente trengtes. Nye kort ble opprettet ved behov. Det ble forsøkt planlegge lengre frem i tid ved å lage kort som innehold mer overordnede oppgaver, noe som ville være med på å lage en større plan for prosjektet.

3.2.2 Sammendrag av iterasjoner

Gjennom bacheloroppgaven ble det planlagt ni iterasjoner totalt. Hver iterasjon varte to uker. På denne tiden skal både løsning være utviklet, og raport fullført.

Prosjektplan (12. januar - 31. jan)

Starten av prosjektet var å lage en prosjektplan. Gruppen brukte hele Januar på å formulere denne prosjektplanen. Gruppen brukte tid på både planlegging av prosjektet og rapporten.

Iterasjon 1 (2. feb - 9. feb)

Første iterasjon gikk med til å begynne å annotere videomateriellet fra NINA. Gruppen dyttet også konseptkoden til det som endte opp å bli den endelige strukturen for databaselaget.

Det ble også brukt tid å på konfigurere utviklingsmiljøet. Dette gjaldt hovedsaklig å sette opp de løsningene som nevnes senere i kapittelet kvalitetssikring.

Iterasjon 2 (10. feb - 23. feb)

Her fortsatte annoteringen av video. Gruppen var ikke klar over hvor tidskrevende arbeidet med annotering kom til å være. Det medførte til at dette ble prioritert en uke ekstra. Det ble også utviklet første prototype av brukergrensesnittet. Gruppen utviklet også første prototype av brukergrensesnittet, slik at det nå kunne vises til oppdragsgiver for tilbakemelding.

Iterasjon 3 (24. feb - 9. mar)

Nå startet utviklingen for fullt. Annoteringen var tilnærmet ferdig og alle begynte å kode på selve løsningen. Gruppen fikk nå lagt grunnlaget ved å begynne implementering av bakenden og arbeidet rundt datamodellene. Sporing ble begynt i denne iterasjonen. En refaktorering av brukergrensesnittet ble også en del av iterasjonen.

Iterasjon 4 (10. mar - 23. mar)

Denne iterasjonen medførte størst endring, og ga mest ny funksjonalitet. Første Maskinlæringsmodellen ble klar, og grensesnittet mellom pakkene begynte å ta form. Spesielt grensesnittet mellom brukergrensesnitt og bakenden var nå på plass. Det ble også implementert funksjonalitet for å eksportere resultat til CSV.

Iterasjon 5 (24. mar - 6. apr)

En ny modell ble nå tatt i bruk. Tidligere modell var ikke like god på grunn av feil ved eksportering. Dette ble jobbet på samtidig som utvikling pågikk. Prosessering av jobber ble nå lagt til. Sammen med en `scheduler` var det nå i mulig til å starte jobber og prosessere flere.

Iterasjon 6 (7. apr - 22. apr)

Gruppen endret bakenden til å returnere mindre informasjon fra klientkall. Fra før ble all data returnert. Når det ble del jobber på et prosjekt, ble datamengden stor slik at det ble nødvendig å implementere en form for å hente kun ønsket data. Sammen med dette ble paginering lagt til. Det ble fort fullt med prosjekt eller jobber på en side. Paginering løste dette med mulighet for å bestemme hvor mange objekter skal vises på en side.

Til brukergrensesnittet ble kommunikasjonen mot bakenden refaktorert til å ta i bruk en application programming interface (API)-klient. Eksisterende kommunikasjon ble håndert på en statisk måte som gjorde det vanskelig å lese koden og samtidig enklere å teste.

Etter en del ytelseproblmenatikk gjennom utviklingen, ble måten å hente ut bilder fra videoene og konvertering av bilder før sending over API.

Iterasjon 7 (3. mai - 11. mai)

Nå begynte alvoret med å skrive i rapporten. Til nå ble en del tid brukt til utvikling og var nå på etterskudd ut fra planen.

Litt finpuss koden ble dyttet opp, men hovedsaklig gikk tiden i å skrive på rapporten.

Iterasjon 8 (12. mai - 20. mai)

Siden gruppen var etter på rapporten ble det nå ikke jobbet videre på løsningen. Tiden ble brukt til å skrive på rapport og lese igjennom innhold produsert.

Kapittel 4

Teori og teknologier

4.1 Teori

4.1.1 Datasyn

Datasyn er et fagfelt der man ønsker at en datamaskin, ut fra bilder og video, skal kunne få en forståelse av verden på lignende måte som et menneske. Det kan være i industri hvor det ønskes å automatisere feildeteksjon på produkter, eller i nyere biler der du har sikkerhetssystemer som gjenkjenner farlige situasjoner og assisterer sjåføren ved å for eksempel bremse om en kryssende forgjenger er detektert.

Dette oppnåes ved å utføre et sett av bildebehandlingsteknikker på bildene for oppnå ønsket resultat. Dette er ofte også utført i kombinasjon med maskinlæring som kommer mer om i avsnitt 4.1.2.

I eksempelet over med fotgjenger trenger datamaskinen å gjenkjenne et menneske. En mulig løsning er å lage en trekk deskriptor av et menneske. En trekk deskriptor er en digital representasjon av et objekt som en datamaskin kan benytte. Denne kan lages ved bruk av for eksempel Histograms of Oriented Gradients(HOG) [6]. Videre kan disse trekkene brukes i en Support Vector Machine(SVM) [7], eller en annen maskinlæringsalgoritme for klassifisering av bilde. Se [8] for videre fordyping i datasyn.

4.1.2 Maskinlæring og dyplæring

Dette avsnittet bygger opp mot en presentasjon av You Only Look Once (YOLO) nettverksarkitektur i slutten av avsnittet, som vil bli benyttet i senere kapittel. Som forkunnskaper for dette gjennomgås det kort om maskinlæring, nevrale nettverk og konvolusjon nevrale nettverk med referanser for videre fordyping.

Maskinlæring og dyplæring er en del av fagfeltet kunstig intelligens. Dette bygger på algoritmer som lærer fra erfaring eller data, istedenfor å programmere inn eksperterfaring i algoritmen, som kan være vanskelig for komplekse system. De kan for eksempel løse problemer innen klassifisering, regresjon eller deteksjon

av unormale hendelser [1, s. 97-101].

Måten algoritmene lærer på kan deles inn i tre kategorier: veiledet læring ("supervised learning") [9, s. 695], ikke veiledet læring ("unsupervised training") [1, s. 142] og forsterkende læring ("reinforcement learning") [9, s. 830].

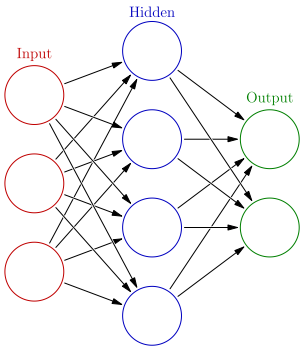
Veiledet opplæring utføres med et datasett hvor hver input har en korresponderende sannhet("ground truth") som modellen bruker under trening. Det betyr at datasettet må annoteres før trening som kan være tidkrevende.

Ved ikke veiledet trening vil algoritmen prøve å trekke ut trekk fra data som den bruker i oppdelingen, som for eksempel gruppering av lignende data. Her trengs det kun rådata og ingen annotering trengs.

Forsterkende læring brukes for opplæring av en autonom agent som mottar informasjon og opererer i sitt miljø [10, s. 367]. Dette kan for eksempel være en sjakkmotor [11] som lærer sjakk ved å spille mot seg selv eller en robot som lærer å utføre en oppgave.

Kunstig nevrale nettverk

Kunstige nevrale nettverk er inspirert fra nevrale nettverk i hjernen. Et framovernettverk("feedforward net") består av flere lag hvor hvert lag består av flere kunstige nevroner, eller *perceptrons*. Dette nettverket er en rettet vektet graf. Et *perceptron* i et lag er koblet til alle *perceptron* i neste lag, se figur 4.1. Hver kobling mellom to *perceptron* har en tilknyttet vekt w_n for hvor stor innvirkning output fra forrige lag har på output fra gjeldende lag.



Figur 4.1: Illustrasjon av et kunstig nevral nettnettverk [12]

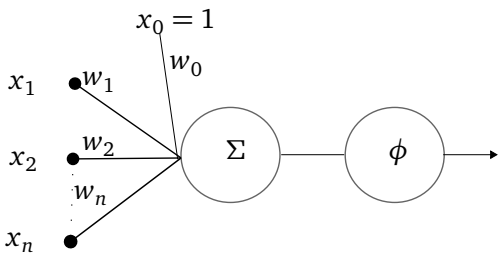
Gitt inngangverdier $\mathbf{x} = (x_1, ..., x_n)$ til perceptron k kan resultatet ut fra k beskrives som summen av den lineære kombinasjonen av produktet av inngangverdi x_i og vekt w_i som er gitt til en aktiveringsfunksjon ϕ som definerer utgangsverdien fra perceptronet. Aktiveringsfunksjonen kan for eksempel være en lineær, logaritmisk eller hyperbolsk funksjon. Illustrasjon av et perceptron kan sees i figur 4.2. Resultat fra k kan uttrykkes som i ligning (4.1)

$$z_k = \phi\left(\sum_{i=0}^{n} w_i \cdot x_i\right) \tag{4.1}$$

hvor ϕ kan være en Sigmoid funksjon [13] som i ligning (4.2).

$$\phi(u) = \frac{1}{1 + e^{-u}} \tag{4.2}$$

Ved å prosessere en inngangsvektor x forover igjennom nettverket vil det gi en prediksjonsvektor y ut fra hva nettverket er trent på.



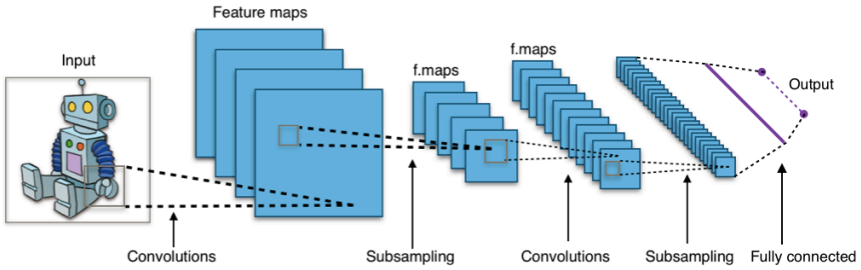
Figur 4.2: Illustrasjon av et perceptron, reprodusert fra [10, s. 87]

Trening av nettverket kan gjøres med veiledet læring. Trening utføres på et datasett hvor hver input har en korresponderende sannhet, slik at nettverkets vekter kan korrigeres under trening ut fra prediksjonen nettverket gjør.

Korrigering av nettveket kan gjøres ved å bruke en *backpropagation*-algoritme. Denne vil lære nettverket ved å prøve å minimere differansen mellom utgangsverdi fra nettverket og korresponderende sannhet ved å korrigere vektene i nettverket. For flere detaljer se [10, s. 97, 14, s. 232].

Konvolusjon nevrale nettverk

Konvolusjon nevrale nettverk [1, s. 328] er et spesiell type nevralt nettverk for prosessering av data som har en rutelignende topologi. Eksempler på dette er bilder som består av et todimensjonalt rutenett med pikselverdier for hver fargekanal i bilde. I motsetning til nevrale nettverk som beskrevet i avsnitt 4.1.2 har ikke hver preceptron kobling til alle i neste lag. Her er det kun koblet til et gitt antall ut fra *kernel*-størrelsen til konvolusjonsoperasjonen. Dette gjør at prosessering av store data blir mer effektiv da det blant annet blir færre vekter i nettverket å kalkulere. Se figur 4.3.

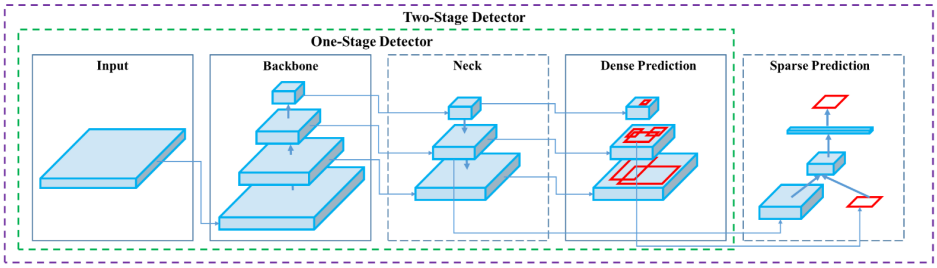


Figur 4.3: Typisk arkitektur av et konvolusjon nevrale nettverk [15]

You Only Look Once (YOLO)

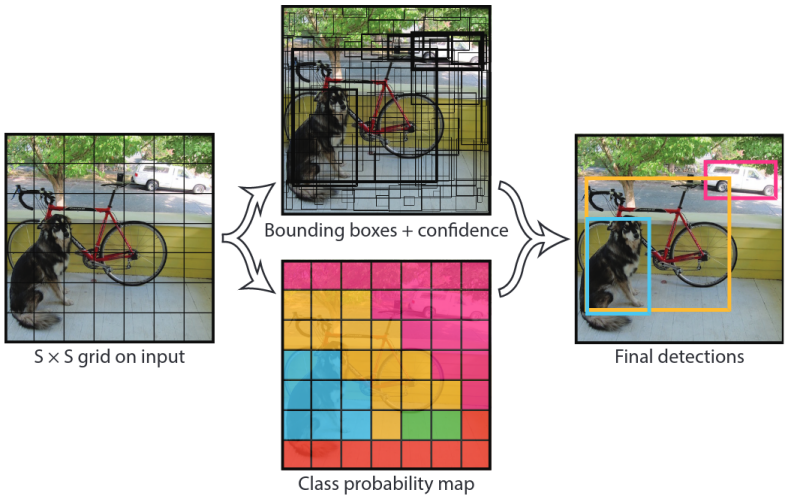
YOLO [16] er en nettverksarkitektur for objektdeteksjon. Det vil si at nettverket vil gi prediksjoner på hvilken type objekter(klassifisering) som finnes i bilde med tilhørende avgrensningsbokser for objektet.

YOLO er en ettstegsdetektor som vil si at nettverket gir avgrensningsbokser og klassesannsynlighet i en evaluering. Se figur 4.4 for en illustrasjon som viser forskjell mellom ett- og tostegsdetektor. Forenklet består nettverkene av en del som gjennomfører konvolusjon på input referert til som *backbone*. Dette er ofte nettverk av typen VGG16, ResNet-50 eller Darknet53. Fra *backbone* blir resultatet overført til *head* av nettverket som er som ofte en *dense* eller *sparse* prediktor. Utformingen her skiller det litt mellom typene etter om nettverket er et ett- eller tostegsdetektor-nettverk. Det er også ofte introdusert et ekstra lag mellom *backbone* og *head* som kalles *neck* av objektdetektoren [17].



Figur 4.4: En stegs vs. to stegs detektor (illustrasjon fra [17])

YOLO finner deteksjoner som et regresjonsproblem. Bildene deles i $S \times S$ nett hvor hver celle i nettet foreslår B avgrensingsbokser og resultat for de boksene. Hver celle gir også en betinget klassesannsynlighet $P(Klasse_i|Objekt)$ [16]. Se figur 4.5 for illustrasjon over nett og avgrensingsbokser i YOLO-modellen. For en detaljert gjennomgang av *YOLOv3* [18] se følgende[1] gjennomgang.



Figur 4.5: YOLO model illustrasjon fra original publikasjon [16]

[1] `https://towardsdatascience.com/yolo-v3-object-detection-53fb7d3bfe6b`, besøkt 08.05.2021

4.1.3 Evaluerings mål

For evaluering av ytelsen til maskinlæringsmodeller brukes diverse statistiske mål. Her defineres noen av de mest vanlige som er brukt videre i rapporten.

De fleste målene bygger på forskjellen mellom sannhet og forslag fra modellen, illustrerer i tabell 4.1.

Tabell 4.1: *Confusion matrix* for klassifisering, illustrerer tp, tn, fn, fp [19]

Sannhet / Forslag	Positive	Negative
Positive	Ekte Positive (tp)	Falske Positive (fp)
Negative	Falske Negative (fn)	Ekte Negative (tn)

Presisjon p er et mål på hvor stor andel tp har av totalt antall foreslåtte positive klassifisering. Alle variablene i ligning (4.3) ligger i første rad i tabell 4.1 [19].

$$p = \frac{tp}{tp + fp} \tag{4.3}$$

Tilbakekalling r er et mål på hvor stor andel tp har av totalt antall sanne positive klassifisering. Alle variablene i ligning (4.4) ligger i første kolonne i tabell 4.1 [19].

$$r = \frac{tp}{tp + fn} \tag{4.4}$$

F_1 verdi gir den harmoniske gjennomsnitt av *presisjon* og *tilbakekalling* som definert i ligning (4.5). Dette gjør at en modell med høy F_1 verdi har god *presisjon* og *tilbakekalling* [19].

$$F_1 = \frac{2 \cdot p \cdot r}{p + r} \tag{4.5}$$

Average Precision(AP) gir gjennomsnitt presisjon for en klasse, som definert i ligning (4.6) [20] hvor n er antall element i klassen.

$$AP = \frac{1}{n} \sum_{i=1}^{n} p_i \tag{4.6}$$

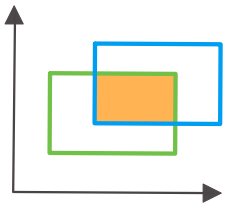
mean Average Precision (mAP) gir gjennomsnittlig presisjon for et sett av klasser, som definert i ligning (4.7) [20] hvor n er antall klasser i settet.

$$mAP = \frac{1}{n} \sum_{i=1}^{n} AP_i \tag{4.7}$$

Disse målene gir et svar mellom 0 og 1, hvor høyere er bedre.

4.1.4 Intersection over Union

Intersection over Union, $\frac{\cap}{\cup}$ er et mål som bestemmer hvor mye overlapp to avgrensingsbokser har [21], se figur 4.6. Avgrensningsboks 1 blir konstruert ut fra linjene x_1, y_1, x_2, y_2, og avgrensingsboks 2 ut fra x_3, y_3, x_4, y_4. For å bestemme snittet må en finne koordinatene som bestemmer den interne rektangelen som defineres med $x_{i1}, y_{i1}, x_{i2}, y_{i2}$. Dette gjøres ut fra ligning (4.8) og ligning (4.9). Snittet bestemmes ut fra ligning (4.11), og union fra ligning (4.10).



Figur 4.6: Blå rektangel er t_n, Grønn er t_{n+1}. Oransje viser overlapp.

$$(x_{i1}, y_{i1}) = (max(x_1, x_3), min(y_1, y_3)) \tag{4.8}$$

$$(x_{i2}, y_{i2}) = (min(x_2, x_4), max(y_2, y_4)) \tag{4.9}$$

$$\cap = (x_{i2} - x_{i1}) \cdot (y_{i2} - y_{i1}) \tag{4.10}$$

$$\cup = (x_2 - x_1) \cdot (y_1 - y_2) + (x_4 - x_3) \cdot (y_3 - y_4) - \cap \tag{4.11}$$

Intersection over Union kan benyttes i sporing av objekt ved å regne overlapp mellom avgrensingsboksene i t_n og t_{n+1} for å bestemme om de tilhører det samme objektet. Det kan også brukes i objektdeteksjon for å bestemme presisjon ved å sammenligne mellom den genererte og sannheten.

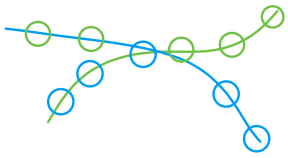
4.1.5 Sporing

Sporing handler om å følge ett objekt over tid, enten for å forutse hvor det kommer til å være, eller for å samle flere datapunkt i ett objekt. Dette kan for eksempel benyttes til å telle individuelle objekt, eller for å anta hvor rom-søppel vil kunne treffe jordkloden.

For å gjøre sporing blir en hypotese laget fra målinger i t_{n-1}. Disse blir så sammenlignet med målinger i t_n. Dersom det er tilstrekkelig overlapp mellom disse punktene blir en hypotese laget for t_{n+1}. Overlappen kan regnes ut med metoder som Intersection over Union, se avsnitt 4.1.4 *Fundamentals of Object Tracking* [22] beskriver metoder som benyttes for å lage hypotesene, spesielt Bayesian og Kalman filter.

SORT er en enkel sanntid(“online”) implementasjon av flerobjektssporing [23]. Her blir datapunkt representert som et areal og en sannsynlighet. Denne benytter hovedsaklig Kalman filter. En videreutvikling av SORT er Deep SORT [24, 25] som i tillegg benytter trekk fra bildet, dette gjør det enklere å finne igjen et objekt som har blitt tildekt. BayesianTracker er en annen implementasjon. Her benyttes Bayesian filter, og har blitt benyttet for sporing av celler [26, 27].

Evaulering



Figur 4.7: Illustrasjon av *mismatch*. Linje viser sannhet, sirkel viser hypotese.

For evaluering benyttes Multiple Object Tracking Accuracy ($MOTA$) og Multiple Object Tracking Precision ($MOTP$) fra [28]. $MOTA$ baserer seg på *false positive*, *mismatch* og *miss* og viser en total feilrate. *False positive* skjer når ingen avgrensningsboks er nære nok hypotesen. Dette kan bety at objektet forsvinner, eksempelvis ved scenebytte eller kutt i datasettet. En *miss* kan tenkes at hypotesen antar objektet ikke vil flytte seg, men den gjør det allikevel. Dersom det skjer en *false positive* impliserer dette også en *miss*. Om to objekt bytter plass som illustrert med figur 4.7 regnes dette som en *mismatch*. $MOTP$ viser den gjennomsnittlige posisjonsfeilraten mellom sannhet og hypotese og finnes i to varianter. En som baserer seg på IoU og den andre på avstand.

Ligning for $MOTP$ vises i 4.12. $d_{i,t}$ er distansen mellom senterpunkene i avgrensingsboksene for hypotesen $h_{i,t}$ og objektet $o_{i,t}$. c_t er antall hypotese- og objektpar for tid t. Resultatet blir mellom 0 og ∞, der 0 er perfekt.

$$MOTP = \frac{\Sigma_{i,t} d_{i,t}}{\Sigma_t c_t} \tag{4.12}$$

Ligning for $MOTP_{IoU}$ vises i 4.13. $IoU_{i,t}$ er overlappen mellom hypotesen $h_{i,t}$ og objektet $o_{i,t}$. c_t er antall hypotese- og objektpar for tid t. Resultatet blir mellom 0 og 1, der 1 er perfekt.

$$MOTP_{IoU} = \frac{\Sigma_{i,t} IoU_{i,t}}{\Sigma_t c_t} \tag{4.13}$$

Ligning for $MOTA$ vises i 4.14. m_t, fp_t og mme_t er alle *miss*, *false positive* og *mismatch*, respektivt for tid t. g_t viser alle objekt i tid t. Resultatet blir mellom 0 og 1, der 1 er prefekt.

$$MOTA = 1 - \frac{\Sigma_t (m_t + fp_t + mme_t)}{\Sigma_t g_t} \tag{4.14}$$

4.1.6 Brukergrensesnitt

Et brukergrensesnitt er det som oftest møter brukeren når et nytt produkt eller en ny løsning skal tas i bruk. Dette grensesnittet kan være digitalt eller fysisk, enkelt eller komplisert. Alt etter hva det prøver å løse. Et digitalt grensesnitt kan lages som et enkelt Command-line interface (CLI)-grensesnitt eller en fullverdig løsning, som for eksempel Microsoft Windows. Ofte er det brukergruppen som bestemmer grensesnittet. Implementering av et brukergrensesnitt kan foregå på mange måter. Mest vanlig er det som en systemapplikasjon, det vil si det brukes biblioteker som følger operativsystemet eller eksterne bibliotek som er kryssplattform eller som en webapplikasjon.

For å lage brukergrensesnittet som en webløsning, benyttes HyperText Markup Language (HTML), Cascading Style Sheets (CSS), og JavaScript. Disse blir brukt

til å bygge opp den semantiske strukturen, angi stil som for eksempel farger, og til slutt gjøre det mer interaktivt. Under følger en kort forklaring på alle tre.

HTML forteller hvordan den bakomliggende strukturen av dokumentet skal være gjennom ulike tagger, som for eksempel <p>. Denne angir bare at dette skal være en paragraf men har ingen direkte endring på grensesnittet. Kontra <img /> som setter inn et bilde og dermed endrer innholdet. Alene utgjør HTML ikke mye om målet er å erstatte et vanlig program. Det er her CSS og JavaScript kommer inn.

For visuelle endringer, benyttes CSS. På samme måte som HTML benyttes det her enkle tagger som forteller om ønsket still. Dette kan være som for eksempel {background−color: red} som forteller at bakgrunnen på element skal være rødt. Paragraf-taggen nevnt over kan nå med CSS bli farget med <p style="background−color: red;">.

På toppen av HTML/CSS kan det benyttes JavaScript. En fordel med JavaScript er for eksempel hvis man trykker på en knapp kan denne endres i sanntid. Uten JavaScript må nettleser sende en oppdatering til tjeneren for så å laste inn nåværende side på nytt. Med JavaScript, kan dette gjøres som en asynkron operasjon. Dette betyr at det kallet som foregår når brukeren trykker på en knapp skjer i bakgrunnen, og påvirker ikke siden brukeren er på. I tillegg benyttes JavaScript til å endre dokumentet i sanntid. For eksempel å legge til bakgrunnsfargen over kan gjøres uten å endre HTML eller CSS direkte, som i utgangspunktet er lagret statisk. Siden endringen ikke blir lagret hos tjeneren, vil den gamle stilen lastet inn neste gang.

Samlet sett med teknologiene over, kan et webløsning erstatte et vanlig program som er skrevet med systembiblioteker. Ved at det utvikles i nettleser gjør det enkelt å lage et grensesnitt raskt uten avhengigheter, som nevnt i innledningen. Resultatet av dette er at løsning er tilgjengelig tvers av de ulike plattformene.

4.2 Teknologier

4.2.1 Maskinlæring rammeverk

Et maskinlæringsrammeverk er et programvarebibliotek som implementerer vanlige konstruksjoner som brukes i maskinlæringsalgoritmer. Dette gir et enklere

grensesnitt for implementering eller utvikling av nye algoritmer innen fagfeltet. Bibliotekene har også ofte enkle grensnitt for vanlige algoritmer som for eksempel nevrale nettverk. Se kodeliste 4.1 for et eksempel av koden for å lage nettverket i figur 4.1.

```
from torch import nn
model = nn.Sequential(
        nn.Linear(3, 4), # input layer
        nn.ReLU(),
        nn.Linear(4, 2), # hidden layer
        nn.ReLU(),
        nn.Linear(2, 2), # output layer
)
```

Kodeliste 4.1: Eksempel bruk av PyTorch

TensorFlow[2] og *PyTorch*[3] er eksempler på rammeverk for å implementere maskinlæring i blant annet programmering språket *Python*. Selv om bibliotekene selv bruker et mer lav nivå språk som *C++* på grunn av ytelse, tilbyr de grensesnitt for forskjellige programmering språk som *Python*.

4.2.2 Python

Python er et tolket, interaktivt og objektorientert programmeringspråk [29]. Det ble startet av *Guido van Rossum* i 1989 og er i dag drevet av *The Python Software Foundation*[4].

Språket tilbyr bruk av moduler, *exceptions*, dynamisk type gjenkjenning, høynivå datastrukturer og klasser. Det kan også bruke andre programmeringsparadigmer som prosedyredrevet eller funksjonell programmering.

Python er portabelt og kan kjøre på tvers av operative system. Det har grensesnitt til systemkall og bibliotek inkludert, i tillegg til et stort antall eksterne biblioteker tilgjengelig gjennom en offisiell pakkebrønn[5].

[2] `https://www.tensorflow.org/`, besøkt 27.04.2021
[3] `https://pytorch.org/`, besøkt 27.04.2021
[4] `https://www.python.org/psf/`, besøkt 01.05.2021
[5] `https://pypi.org/`, besøkt 01.05.2021

4.2.3 Pytest

Pytest[6] er et testrammeverket som består av et Python-bibliotek og et kjørbart skript. Skriptet ser etter filer i en `tests/`-mappe, og innenfor der ligger det Python-filer med prefiks `test_`. Innenfor en slik test ligger det igjen funksjoner som også har prefiks `test_`. En test må bestå av minst en `assert`. Dette er en funksjon i Python tar inn en kondisjon, og kaster en *exception* dersom denne ikke er sann.

4.2.4 Poetry

Poetry[7] er et verktøy for Python-utvikling. Poetry gjør det enkelt å pakke løsningen for distribusjon, og for å handtere løsningens avhengigheter. Disse deles opp i to typer, utvikling og vanlig. Utviklings-avhengigheter kan være spesifikke program utviklerene trenger, som formateringsverktøy eller debugger. Vanlige avhengigheter kan være bibliotek som trengs for å kjøre løsningen. Når en avhengighet legges inn i Poetry kan dette være fra den offisielle Python-pakkebrønnen eller lenke til et annet Poetry prosjekt.

4.2.5 Flask

Flask er et webrammeverk for Python. Som et "mikrorammeverk" inneholder det ikke mange funksjoner og tilbyr kun det minimale for å sette opp en nettside. Flask inneholder en enkel webtjener[8] som tilbyr nettsidene og en template-motor[9] for å støtte dynamisk innhold. En template-motor gjør det mulig å lage dynamiske sider som kan hente informasjon fra en eksterne kilder.

Selv om Flask er begrenset med hvilke funksjoner som leveres, eksisterer det utvidelse som kan tilby ønsket funksjonalitet. Det eksisterer et stort miljø rundt rammeverket. Dette gjør det enkelt å finne utvidelser som andre har utviklet. Eksempel på en slik utvidelse er flask-paginate[10].

4.2.6 Tailwind CSS

Tailwind CSS er et CSS-rammeverk, videre kalt kun Tailwind. Rammeverket benytter seg av egendefinerte klasser som man legger rett inn i HTML. Fordelen med Tailwind er at man legger til enkle klasser som gir ønsket effekt. Til kontrast med vanlig CSS lager man ofte kun én klasse som kan definere hele komponenten. Men med Tailwind brukes mange klasser som hver definerer kun én enkel stil.

Rammeverket har en rekke valg definert som standard, men disse kan både fjernes eller endres ved hjelp av en konfigurasjonsfil. I utgangspunktet definerer Tailwind sin egen standard, men den kan endres til å passe eget bruk.

---

[6] `https://docs.pytest.org/en/6.2.x/`, besøkt 13.05.2021
[7] `https://python-poetry.org`, besøkt 19.05.2021
[8] `https://palletsprojects.com/p/werkzeug/`, besøkt 10.05.2021
[9] `https://palletsprojects.com/p/jinja/` besøkt 10. 05.2021
[10] `https://flask-paginate.readthedocs.io/`, besøkt 10.05.2021

4.2.7 FastAPI

FastAPI[11] er et grensesnitt-rammeverk for Python. FastAPI bygger på Starlette[12] som er et Asynchronous Server Gateway Interface (ASGI)-rammeverk. FastAPI benytter også Pydantic[13], en dataanalyse- og valideringspakke basert på Python typehinting, for validering av data. Basert på åpne standarder for grensesnitt som OpenAPI[14] gir FastAPI også automatisk dokumentasjon av grensesnittene.

FastAPI gir en enkel måte å definere endepunkt i ren Python-kode[15] med automatisk type validering og avhengighet injeksjon. Det er også mye tilgjengelig funksjonalitet igjennom Starlett eller "plug-ins".

4.2.8 SQLAcademy

SQLAcademy er et bibliotek for å abstrahere vekk operasjoner mot databaser. Det kan kobles opp mot det fleste kjente databasetjenere som SQLite [16] og PostgreSQL [17] [30, 31]. Databasestrukturen kan opprettes ved bruk av en Object Relational Mapper (ORM) i kode, uten å måtte røre SQL-språket [18]. ORM handler om å kople kodens objekt opp mot en database, slik at en kun definerer sine objekter [32]. Dette gjør til at en ikke trenger bruke mye tid på selve databasedesign ved eksempelvis Enhanced Entity-Relationship (EER) [33].

SQLAcademy benyttes også for å kople opp mot en database ved hjelp av en Engine [19]. Videre kan en Session [20] opprettes som blir måten du videre interakterer med databasen.

[11] `https://fastapi.tiangolo.com`, besøkt 12.05.2021.
[12] `https://www.starlette.io/`, besøkt 12.05.2021
[13] `https://github.com/samuelcolvin/pydantic/`, besøkt 12.05.2021
[14] `https://www.openapis.org/`, besøkt 12.05.2021
[15] `https://fastapi.tiangolo.com/tutorial/first-steps/`, besøkt 12.05.2021
[16] `https://sqlite.org/index.html`, besøkt 12.05.2021
[17] `https://www.postgresql.org/`, besøkt 12.05.2021
[18] `https://en.wikipedia.org/w/index.php?title=SQL&oldid=1021733415`, besøkt 17.05.2021
[19] `https://docs.sqlalchemy.org/en/14/core/connections.html#sqlalchemy.engine.Engine`, besøkt 17.05.2021
[20] `https://docs.sqlalchemy.org/en/14/orm/session_api.html#sqlalchemy.orm.Session`, besøkt 17.05.2021

Kapittel 5

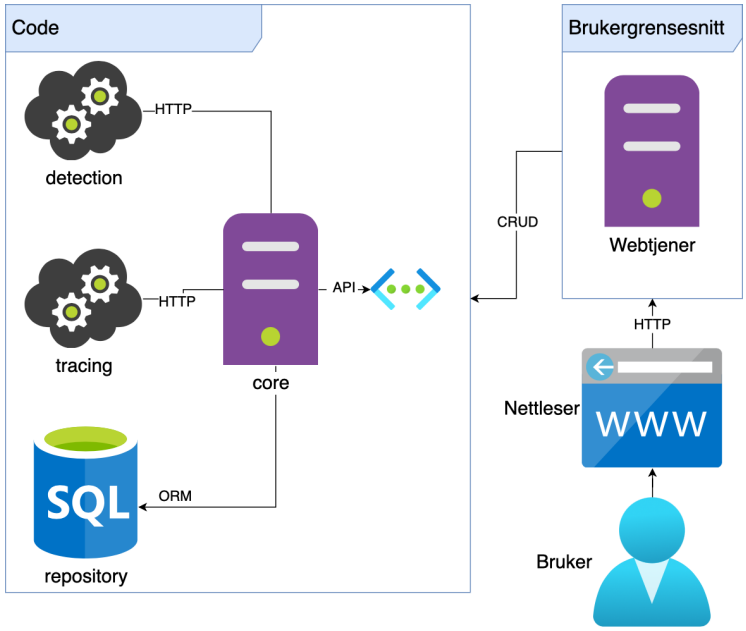
Design og implementasjon

Den overordnede struktur og arkitekturen til løsningen var noe fastsatt av krav i avsnitt 2.2. Det må legges til rette for at løsningen skal kunne deles opp og kjøres på forskjellige enheter og lokasjoner. Derfor ble det undersøkt etter arkitekturer som egner seg for distribuert drift. Arkitekturer som ble vurdert var *pipe and filter* [34], lagdeling [35], tjenesteorientert [36] og mikrotjeneste [37].

Det ble valgt å implementere løsningen ved å bruke en mikrotjeneste ("microservices") arkitektur. Hvert domene som identifisert i avsnitt 2.1.2 ble implementere som sin egen mikrotjeneste som kan kjøre uavhengig av de andre. Tjenestene kommuniserer igjennom bruk av Hypertext Transfer Protocol (HTTP). Oversikt over struktur og kommunikasjons avhengighet mellom tjenestene kan sees i figur 5.1.

Det tre domenene definert i kapittel 2 ble delt inn i følgende Python-pakker:

- **Detection** er hvor objektdeteksjon gjennomføres. Bildedata blir sent inn over HTTP-grenesnitt fra *Core*, og tilbake kommer deteksjoner. Les mer i kapittel 6.
- **Tracing** samler deteksjoner til objekt. Her blir deteksjonene sent inn over HTTP-grensesnitt fra *Core*, og tilbake kommer objekt. Les mer i kapittel 7.
- **Core** er applikasjonens kjerne. Her blir videofiler klargjort for prosessering, og analysert data lagret. Les mer i kapittel 8.
- **UI** er brukerens syn inn til applikasjonen. Her opprettes prosjekt og jobber, og prosessering igangsettes av bruker. Les mer i kapittel 9.



Figur 5.1: Overordnet oversikt av arkitektur og forhold mellom moduler i løsningen.

Kapittel 6

Objektdeteksjon

Dette kapittelet er strukturert etter IMRAD-modellen. Her vil problemstillingen med å detektere objekter i video bli utforsket og dokumentert.

6.1 Introduksjon

For å detektere og klassifisere artene i videoen trengs det en metode for å detektere objekter av interesse, for så å klassifisere disse. Dette kan for eksempel løses ved hjelp av tradisjonelle datasynsmetoder eller ved dype læringsmetoder.

En mye brukt teknikk for å detektere bevegende objekt i scener med statisk kamera er "bakgrunn subtraksjon" [38]. Fra de detekterte objektene kan det brukes håndlagede trekk deskriptorer som Scale-Invariant Feature Transform (SIFT) [39] og Speeded Up Robust Features (SURF) [40]. Trekkene er så brukt for å lære en Support Vector Machine (SVM) eller K-Nearest Neighbors (KNN) algoritme til å skille mellom artene som så brukes til å predikere på nye bilder. Ved denne type løsning må de viktige trekkene av artene velges under utvikling. Deretter blir de brukt til å klassifisere objektene. I motsetning til ved dyp læring, der nettverket lærer trekkene ut fra treningsdata direkte. En fordel med en datasyn tilnærming er at mengden treningsdata som trengs er mindre enn ved til en dyp læring tilnærming. Med datasyn er det lettere å forklare hva som skjer enn med dyp læring, hvor det er mer som en sort boks.

Dyp læring er et fagfelt som er i rask utvikling. Nye og forbedrede teknikker publiseres jevnlig. En mulig tilnærming for å løse problemet med dyp læring er typisk å bruke et nettverk basert på konvolusjonalt nevrale nettverk som introdusert i avsnitt 4.1.2. Disse nettverkene finner deteksjoner av objekter i bildene. Dette kan for eksempel være nettverk av typen Region-based Convolutional Neural Networks (R-CNN) [41], You Only Look Once (YOLO) [16] eller Single shot multibox detector (SSD) [42]. Nettverkene finner, avhengig av hva de er trent på, hvilke type objekter som er i bilde og hvor de er vist med for eksempel en avgrensningsboks. Dyp læring trenger mindre ekspert analyse og justeringer enn datasyn ved å benytte store datamengder [43].

Tabell 6.1: Sammenligning av resultater fra publisering av nettverksarkitekturer. Tall i sammenligning fra [17] foruten for *YOLOv5* [44].

Nettverk	Størrelse	AP	FPS
EfficientDet-D0 [45]	512	33.8%	62.5
EfficientDet-D1 [45]	640	39.6%	50.0
YOLOv3-Darknet-53 [18]	416/608	40.6/42.4%	53/45.5
YOLOv4-CSPDarknet-53 [17]	512	43.0%	83
RetinaNet-ResNet-50	640	37.0	37
YOLOv5m5 [44]	640	44.5%	370 *(2.7ms)
YOLOv5l5 [44]	640	48.3%	263 *(3.8ms)

Notat: * FPS kalkulert fra inferens tid i ms, $FPS = \frac{1}{tid/1000}$. Metodene for ytelses testing kan avvike mellom publikasjoner, så tallene her må sees på som veiledende da ingen verifisering er gjennomført.

Ut fra dette er det valgt å følge en dyp læring løsning på problemet. Dette har også vist seg å prestere bedre enn tradisjonelle datasynløsninger. Ulempene er at det er ytelseskrevende å trene nettverket i tillegg til at det er nødvendig med tilstrekkelig og variert treningsdata [43]. Noe som var et fagfelt gruppen ønskte å utforske.

6.2 Metode

6.2.1 Valg av nettverks arkitektur

Ved valg av nettverk ble det lagt vekt på at løsningen bør kunne prosessere data i nær sanntid, eller raskere, for at tiden til prosessering skulle bli minst mulig. Ut fra resultater publisert i tidligere sammenligninger er tabell 6.1 satt opp med resultat og ytelse oppnådd for en modell trent på datasettet Common Objects in Context (COCO).

En annen vurdering er hvor lett det er å bruke nettverket i et maskinlærings-rammeverk. Mange publiserte metoder er ikke utviklet til å være produksjonsklare og disse ville tatt mye tid for å implementere nettverket i et rammeverk. Det igjen ville gitt muligheter for feil i implementeringen ut fra den informasjonen som er publisert. For gruppen er dette et viktig kriterium siden bare en i gruppen har erfaring med maskinlæring og datasyn. Ut fra dette kriteriet ble RetinaNet, Faster R-CNN og You Only Look Once (YOLO) identifisert som å være bra utviklede i rammeverk som *TensorFlow* og *PyTorch*.

Gitt tilgjengelighet i rammeverk og ytelse ble det valgt å bruke en versjon av YOLO for prosjektet. Et publisert nettverksdesign basert på YOLO er *YOLOv4* [17]. Denne bygger på tidligere publisert versjon YOLOv3 [18]. Det finnes også en YOLOv5 [46] som bygger også på YOLOv3, men er implementert i maskinlærings-

rammeverket *PyTorch*[1] isteden for rammeverket *Darknet*[2], som de tidligere versjonene er basert på. Med *Darknet* skrives nettverks implementasjonen i programmeringsspråket C, hvor treningen utføres ved å kjøre et kompilert treningsprogram. Vektene fra treningen kan implementeres i egen løsning ved hjelp av programvarebiblioteket *OpenCV* sin dyp lærings modul. YOLOv5 er under aktiv utvikling, men har ikke publisert noen resultater. Det ligger derimot referanseindeks for model trent på datasettet COCO på kodebrønnen til prosjektet [44]. På bakgrunn av noen forkunnskaper med *PyTorch*, enkel trening, bruk av modellen med hjelp fra *TorchHub*[3] og ytelses egenskapene ble YOLOv5 valgt for prosjektet.

6.2.2 Datasett

Oppdragsgiver bisto med 171 minutter(21 GB) med utvalgte og sortert videoklipp av hver art. Videoene hadde en oppløsning på 1920 × 1080 med 25 bilder i sekundet. Klippene var av artene som ble introdusert i kapittel 1, i varierende lys og siktforhold. Disse klippene ble brukt til å lage et datasett for bruk i prosjektet.

Verktøy

For å annotere videoene ble verktøyet Computer Vision Annotation Tool (CVAT) [47] brukt til å markere artene med korrekt avgrensningsboks og klasse, se brukergrensesnittet i figur 6.1.

For hver video annotert ble det generert og eksportert et datasett fra CVAT. Alle datasett ble filtrert for bilder uten annotering og kombinert til et datasett ved bruk av verktøyet Dataset Management Framework (Datumaro) [48]. Det ble også brukt for å splitte datasettet inn i trening, validering og testsett. Vertøyet bevarer klasse distribusjonen fra det samlede datasettet[4].

Et datasett i *YOLO* format ble eksportert fra Datumaro og deretter manuelt endret til å følge *YOLOv5* sitt forventede oppsett[5].

Annoteringsregler

Det ble bestemt noen generelle regler for annotering av datasettet for at det skulle bli mest mulig likt annotert på tvers av medlemmene. En regel var at avgrensningsboks ble laget rundt hele arten, inkludert finner på fiskene. Annotering startet når hode til arten er inne i bilde og sluttet når mer en halve fisken var ute av bilde. Om arten forsvant eller kom inn i bildet fra bakgrunnen, startet eller stoppet annoteringen når arten sin kontur forsvant.

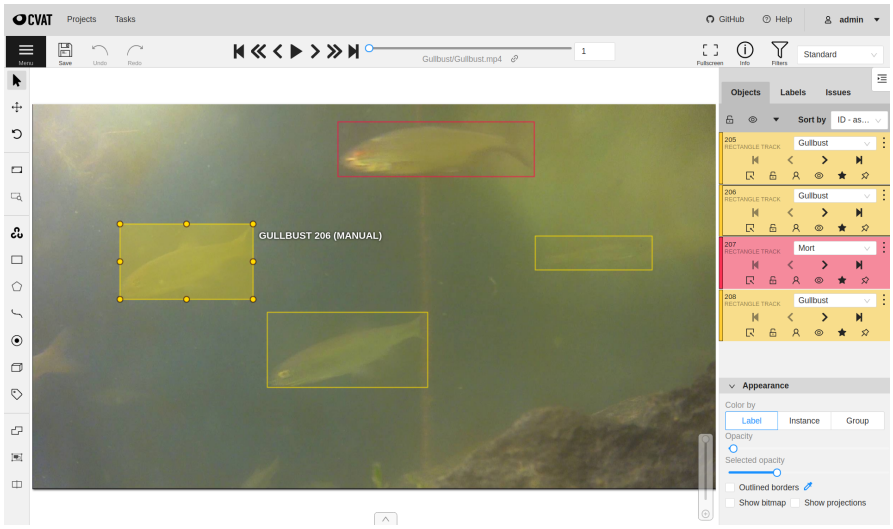
[1] `https://pytorch.org/`, besøkt 22.02.2021.

[2] `https://github.com/pjreddie/darknet`, besøkt 22.02.2021.

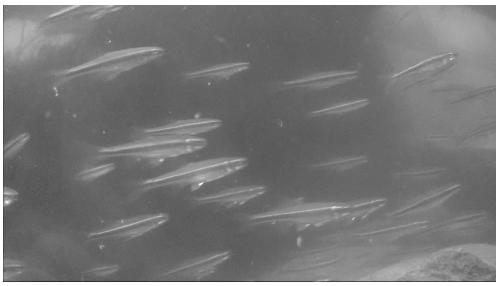
[3] `https://pytorch.org/hub/`, besøkt 22.02.2021.

[4] `https://github.com/openvinotoolkit/datumaro/blob/develop/datumaro/plugins/splitter.py#L459`, besøkt 23.02.2021

[5] `https://github.com/ultralytics/yolov5/wiki/Train-Custom-Data/0e7383e1c729d50d9830ed56f01174a93ce02db#3-organize-directories`, besøkt 23.02.2021



Figur 6.1: Computer Vision Annotation Tool (CVAT)



Figur 6.2: Eksempel på fisk i stim.

Noen av artene svømmer ofte i stim, noe som gjør det utfordrende og tidkrevende å annotere alle individene over tid i videoene, se figur 6.2. For å spare tid, ble det bestemt at det var nok å fullstendig annotere noen rammer fra videoen. Dette kan være noen bilder av en stim, eller en kortere periode av videoen. Konsekvens av dette er at disse videoene ikke kan bli brukt for testing av sporingen i løsningen siden det mangler annotering som kan brukes som referanse.

Datasett oppdeling

Datasettet ble delt opp i et trening, validering og testsett. Der 70% ble brukt til trening og resterende delt mellom validering og testsett. Fordelingen mellom klassene ble opprettholdt i fordelingen.

6.2.3 Håndtering av ubalansert datasett

For noen av artene var det veldig lite data, mens på andre arter var det mye data. Dette gjorde at datasettet ble ubalansert, som måtte tas hensyn til.

En løsning på problemet er å generere eller finne mer data til å inkludere i datasettet. En metode for å generere mer data ut fra datasettet som allerede eksisterer, er å manipulere bildene for å skape et nytt bilde. Noe som ofte kalles *data augmentation*. Dette kan være operasjoner som rotasjon av bilde, speilvende bilde

Tabell 6.2: VM spesifikasjoner

Flavor Name	gpu.v100.8G
RAM	90GB
VCPUs	8 @ 2.6Ghz (Intel Xeon Processor (Skylake, IBRS))
Disk	40GB
Image Name	Ubuntu Server 20.04 GRID CUDA 11.0
GPU	NVIDIA GRID V100D-8Q

eller endre fargetonen i bilde. Operasjonen kan utføres på forhånd i datasettet, eller det kan utføres under trening gjennom en serie av tilfeldige transformasjoner. Øking av data på denne måten har vist seg å kunne forbedre nøyaktigheten av klassifiseringen [49] fra modellen.

En anbefalt metode for ubalanserte datasett er å benytte oversampling av klasser som er i minoritet [50]. De anbefaler oversampling av minoritetklassene inntil ubalansen er eliminert, og for best resultat sette terskel for å kompensere for sist klasse sannsynlighet.

6.2.4 Implementation og trening av nettverk

Treningen ble utført på en Virtual Machine (VM) i *OpenStack*, "skyen" til NTNU. En maskin med Graphics Processing Unit (GPU) ble brukt, se spesifikasjoner i tabell 6.2.

For trening av modellen ble kodebrønnen for YOLOv5 klonet til maskinene der treningen ble gjennomført. Ved bruk av treningsrutinen[6] i YOLOv5 ble prosjektets datasett brukt for videre trening av en model trent på COCO gjennom å kjøre kommandolinje instruksjonen i kodeliste 6.1.

```
python train.py --img 640 --batch 19 --epochs 100 --data \
/mnt/fast/dataset/dataset.yaml --weights yolov5m.pt --hyp hyp.yaml \
--image—weights
```

Kodeliste 6.1: Kommando for å starte trening

Hvor hyp.yaml definerer hyperparameter som skal brukes i treningen. Filen definerer innstillinger som læringrate, *loss gain* og *data augmentation* oppsett som skal brukes. Mulige metoder for å øke datamengden ved å gjøre endre på bildene under trening er:

- Endring i fargetone, metning eller valør
- Rotasjon
- Transformasjon

[6] `https://github.com/ultralytics/yolov5/wiki/Train-Custom-Data/6925a1e83e3e5e307cca32b526e230208b77d88b#5-train`, besøkt 10.02.2021.

- Skalering
- Skjær
- Perspektiv
- Speilvende opp/ned eller venstre/høyre
- Lage mosaikk av bildene
- Blande bilder

Alle disse metodene ble til varierende grad brukt, foruten skjær, perspektiv og blanding. Standard innstillinger for *YOLOv5* og brukte innstillinger er gitt i vedlegg D.

For bruk av modellen i løsningen ble det implementert et enkelt web application programming interface (API). Programmet kan gjøre spørringer av bilder igjennom et application programming interface (API)-kall. APIet vil svare med predikasjoner og sannsynligheter for hvert detektert objekt i bildet.

6.3 Resultat

6.3.1 Datasett

Endelig datasett består av 85194 bilder med 161753 annoteringer, hvor fordelingen mellom klassene er som vist i tabell 6.3. Av disse er 14129 annoterte nøkkelpunkt og resterende er verifiserte interpoleringer. Det ble også lagt til 461 bakgrunnsbilder uten fisk med varierende lys og sikt forhold. For å annotere og ferdigstille datasettet er det timeført 100 timer med arbeid.

Tabell 6.3: Fordeling mellom klassene i datasettet.

Klasse	Antall annoteringer	Distribusjon
Abbor	20397	0.125
Brasme	334	0.002
Gjedde	54774	0.336
Gullbust	2064	0.013
Mort	8363	0.053
Rumpetroll	12274	0.075
Stingsild	423	0.002
Vederbuk	23362	0.143
Ørekyt	40864	0.251

6.3.2 Trening

De første øktene med trening ble utført med et datasett som inneholdt feil, som diskutert i avsnitt 6.4.1. Resultatet fra disse er ikke tatt med i denne seksjonen.

Oversikt over treningsøkter på korrekt datasett kan sees i tabell 6.4 med resultater visualisert i figur 6.3. *Confusion matrix* for økt nummer 1 kan sees i figur 6.4.

Modellene i økt 1 og 2 tar lengre tid å trene på grunn av de bruker større bildestørrelse.

Tabell 6.4: Trenings parameter brukt for forskjellige treningsøkter

Økt	Modell	Størrelse	Epoker	Batchstr.	Hyperparam.
1	YOLOv5m6	720	300	12	Standard
2	YOLOv5m6	1280	200	4	Justert *
3	YOLOv5m5	640	300	19	Justert *
4	YOLOv5s5	640	300	32	Justert *

Notat: * som definert i avsnitt 6.2.4.

6.3.3 Ytelse av modell

De trente modellene var testet for hvor fort modellene kan utføre prediksjoner på testdatasettet, resultatet kan sees i tabell 6.5.

Tabell 6.5: Ytelses målinger av trente modeller pr. bilde med størrelse 640×640 på VM med GPU. Gjennomsnitts tid per bilde av en *batch* med størrelse 32.

Modell	Tid per bilder i *ms*
1.YOLOv5m6	10.3
2.YOLOv5m6	10.7
3.YOLOv5m5	7.8
4.YOLOv5s5	6.7

6.3.4 Implementation

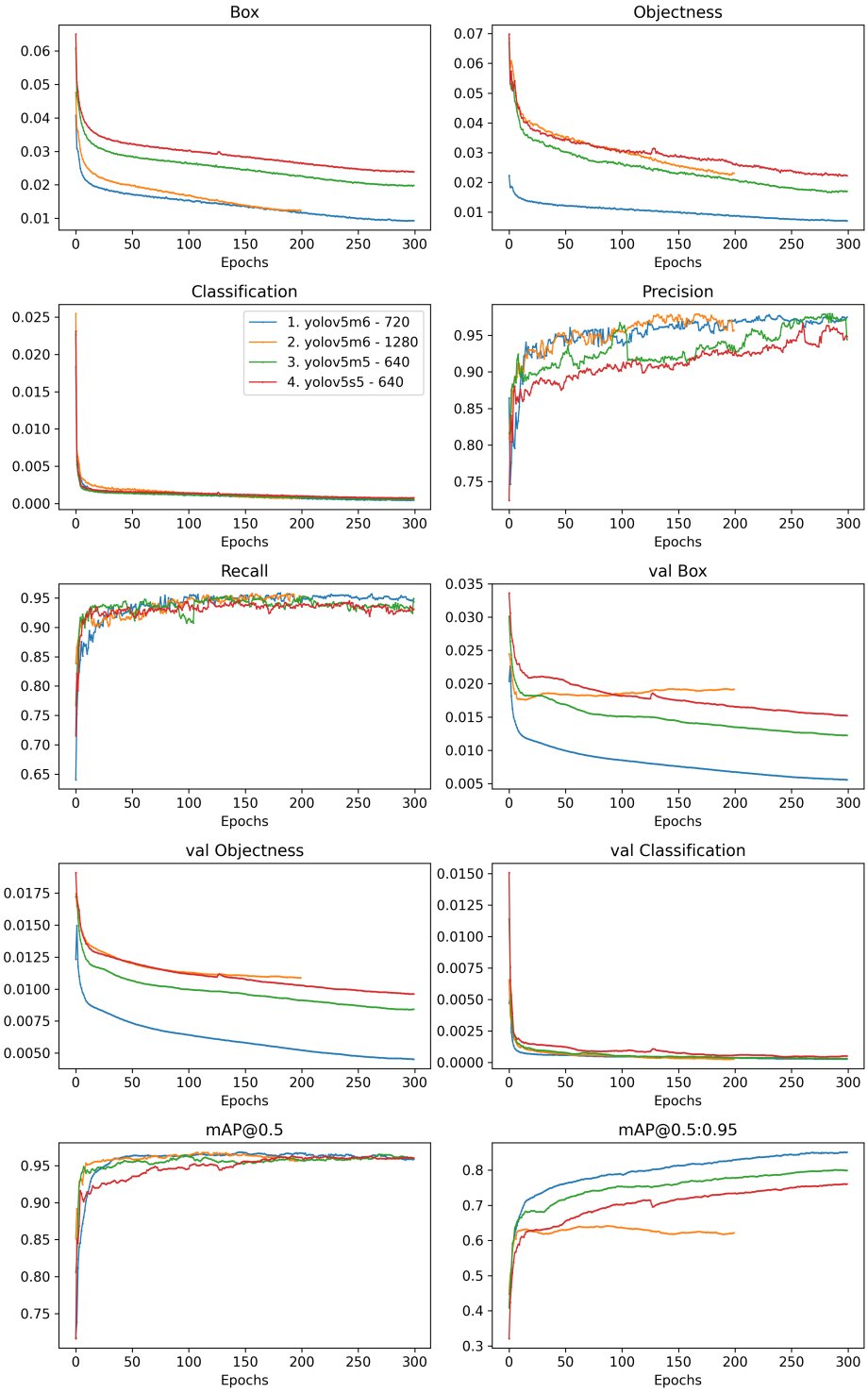
Deteksjons løsningen ble implementert som en mikro-tjeneste i sin egen Python pakke. Denne kan kjøres opp separat fra de andre delene av løsningen fra en kommandolinje.

APIet for bruk av modellen ble implementert ved bruk av FastAPI rammeverket i Python. Se figur 6.5 for *OpenAPI* dokumentasjon av grensesnittet. Den tilbyr to endepunkt løsningen kan gjøre spørringer på. Et for å få en oversikt over modeller som er tilgjengelig, og et annet for å kjøre deteksjon på bilder med en gitt modell.

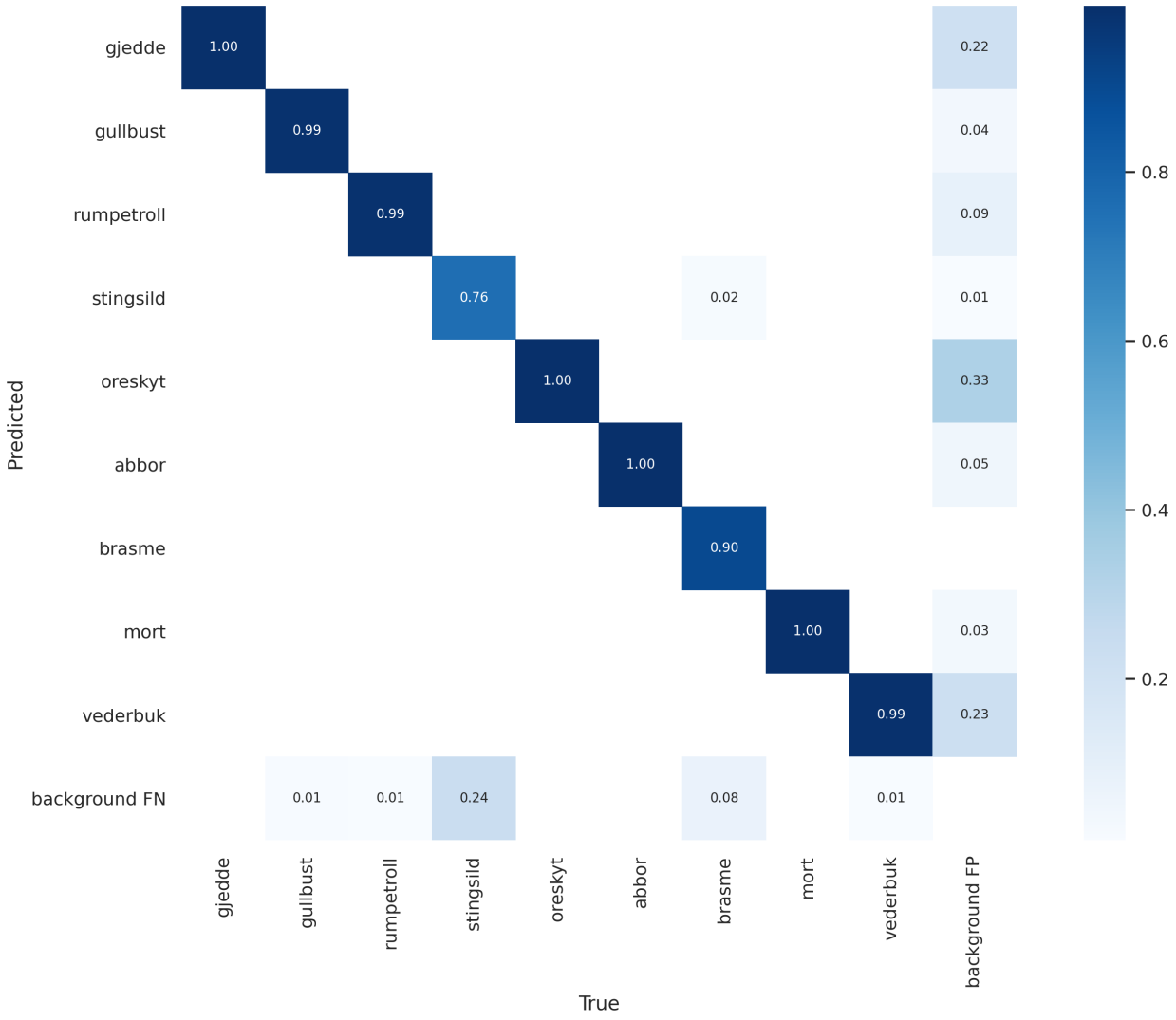
6.4 Diskusjon

6.4.1 Datasett

I starten ble hele videoer annotert. Dette resulterte i store mengder data, men tok også lang tid. Derfor ble det etterhvert bestemt at annotering av hele videoene ikke var fornuftig bruk av tiden. Annotering av en art ble utført helt til en hadde tilstrekkelig data til trening.



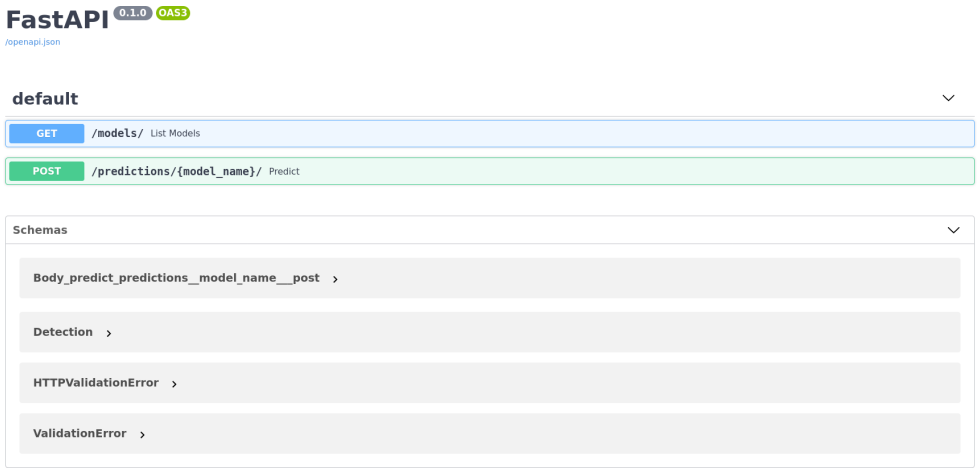
Figur 6.3: Treningsresultat for treningsøktene i tabell 6.4. Økt 1,3 og 4 trente i 300 epoker, mens økt 2 trente i 200 epoker.



Figur 6.4: *Confusion matrix* for treningsøkt nr. 1, modell YOLOv5m6

Det viste seg etter hvert at kombineringen av datasettene eksportert fra CVAT med Datumaro ikke hadde fungert som forventet. Grunnen var at filnavn var for like mellom datasettene. Dette gjorde at Datumaro kombinerte annoteringer fra filer med samme filnavn, som igjen resulterte i at treningen ga dårlige resultater. Etter dette problemet ble identifisert ble det brukt en del tid på å kombinere datasettene på nytt, og verifisere at det nye datasettet var korrekt. Datasettet ble verifisert ved å tegne annoteringer på bildene og vise et bilde ved gitte mellomrom, se skript for verifisering i vedlegg O.

Det ble prøvd å sette opp CVAT i skyen slik at alle medlemmene hadde tilgang til samme instans. På grunn av diverse problemer med oppsett av CVAT, ble



Figur 6.5: Deteksjon API OpenAPI dokumentasjon

løsningen erstattet med at hver enkelt installerte det lokalt.

Bruken av CVAT viste seg å være problematisk for noen av medlemmene. En i gruppa opplevde at Docker-container som kjørte CVAT sluttet å virke med jevne mellomrom, og derav mistet annotert data som ikke var lagret. For å unngå å miste data ved fremtidige problem ble automatisk lagring hvert andre minutt aktivert i innstillingene. Et annet problem som oppstod sent i annoterings perioden hendte etter et medlem hadde eksportert ferdig datasett, og stoppet CVAT Docker-containeren. Den lagrede dataen i CVAT ble enten slettet eller korrupt. Derfor var det ikke mulig å gå tilbake å gjøre endringer. Foruten ved å importere eksportert datasett, eller å starte annotering på nytt. Hadde CVAT blitt satt i skyen som tenkt med rutiner for lagring av sikkerhetskopier kunne muligens disse problemene blitt unngått. Det viste seg å være problematisk som diskutert over.

Det ble også søkt etter mer data for artene *brasme* og *stingsild* på grunn av lite data. Oppdragsgiver ble også forespurt om de hadde mer data for disse artene, noe de ikke hadde. Det ble funnet bildedata med en åpen lisens hos *Global Biodiversity Information Facility*[7]. Ved hjelp av deres API ble bildene lastet ned og annotert i CVAT. Deretter implementert inn i datasettet. Med disse bildene ble antallet for disse artene økt til henholdsvis 334 og 423 annoteringer, men mange av bildene var av fisk på land og ikke i et undervannsmiljø.

6.4.2 Trening

Fra resultatene kan det sees i figur 6.4 at det er problemer med falske positive deteksjoner for enkelte av artene i bakgrunnen. Dette kan være et resultat av at annoteringen av disse artene kanskje ble startet eller sluttet for seint, hvor arten var for lik bakgrunn. Modellen viser også at den klarer bra å detektere artene i bilde, foruten problemer mellom brasme og stingsild. Dette er som forventet da

[7] `https://www.gbif.org`, besøkt 10.03.2021

disse klassene er det med minst data som diskutert i avsnitt 6.4.1. Av de ekstra bildene som ble lagt til av disse artene er også mesteparten bilder tatt av arten på land, så de kan inneholde trekk som ligner.

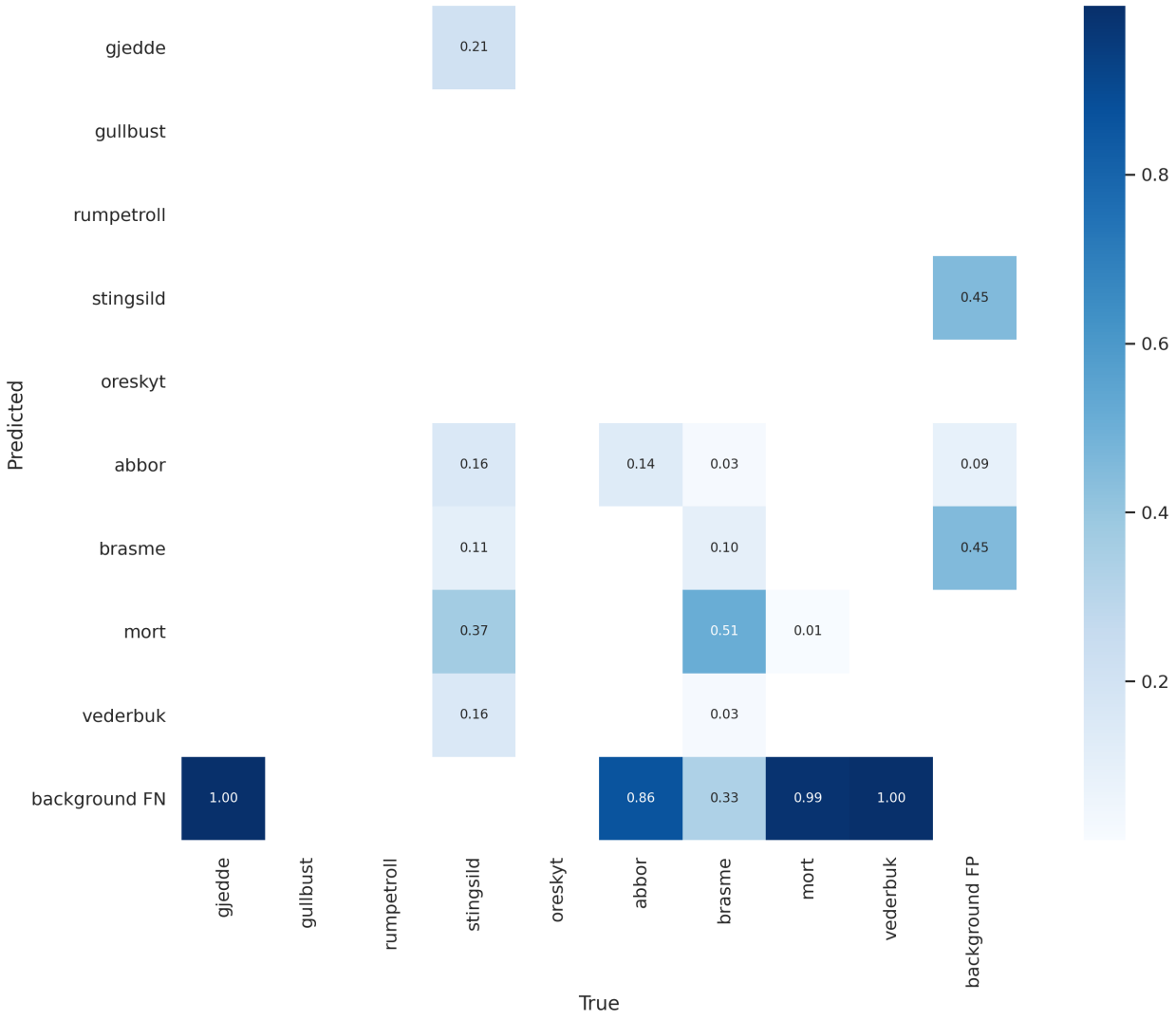
Av treningsøktene i tabell 6.4 er det økt nummer en som når høyest mean Average Precision (*mAP*) ved $IoU = 0.5 : 0.95$ etter 300 epoker som vist i grafen i figur 6.3. Dette er nok på grunn av at det er mindre endringer i bildene mellom hver trenings epoke, så modellen gjenkjenner lettere treningsdata. Det kan tenkes at denne modellen vil være mindre generisk, da bildene på denne foreksempel ikke ble tilfeldig rotert som det andre ble. Dette har det ikke blitt satt av tid for å undersøke nærmere. Resultatene fra alle øktene når målet $mAP_{IoU=0.5} = 0.6$ satt i avsnitt 1.3.2 etter rundt 10 epoker og alle øktene virker som konvergerer mot en $mAP_{IoU=0.5} = 0.95$ etter rundt 200 epoker. Målet som var satt under prosjektplanleggingen kan vurderes som at det var satt litt konservativt. Av øktene er nummer en den som gir best presisjon og tilbakekalling. Der av får denne også best $mAP_{IoU=0.5} = 0.959$. Ut fra dette er denne modellen brukt videre under utviklingen av løsningen.

Siden bildene i treningsett og valideringsett kommer fra de samme videoene, er muligheten for at disse er for like til å detektere overtrening(*overfitting*) [14]. Figur 6.3 viser lite tegn til at valideringsett *Box*, *Objektness* eller *Classification* tap starter å øke. Det er mulig at det ikke er trent lenge nok for at disse vil starte å øke.

Ved hjelp av oppdragsgiver ble videoer identifisert for noen av artene på internett. Noen bilder fra disse ble dratt ut og annotert mot slutten av prosjektet. Disse ble brukt som et ekstra testsett for å sjekke modellen. Dette datasettet besto av 37 bilder med 179 annotering-er av fem av artene. Test på disse viste at modellen ikke gir en god generalisering av klassene i et annet miljø, se *confusion matrix* i figur 6.6, figur 6.7 og tabell 6.6. Fra de artene som eksisterer i dette testdatasettet, blir de fleste feil detektert som bakgrunn(falsk negative). Dette selv om resultat fra validering- og testsett gir gode resultat som ville vært innenfor målene satt i kapittel 1. På bakgrunn av dette sees det at mer tid burde vært brukt på denne delen for å få et bedre result, men dette ble oppdaget for seint i utviklingen til at dette kunne vurderes. Det vil være behov for å videre undersøke muligheter for å trene en modell som gir bedre generalisering.

Tabell 6.6: Resultat fra ekstra testdatasett på modell fra økt 4

Klasse	**P**	**R**	**mAP@.5**	**mAP@.5:.95:**
alle	0.738	0.145	0.117	0.0767
gjedde	0.6	0.2	0.212	0.153
abbor	0.81	0.0303	0.0524	0.0414
brasme	0.281	0.474	0.204	0.121
mort	1	0.0189	0.114	0.0679
vederbuk	1	0	0	0

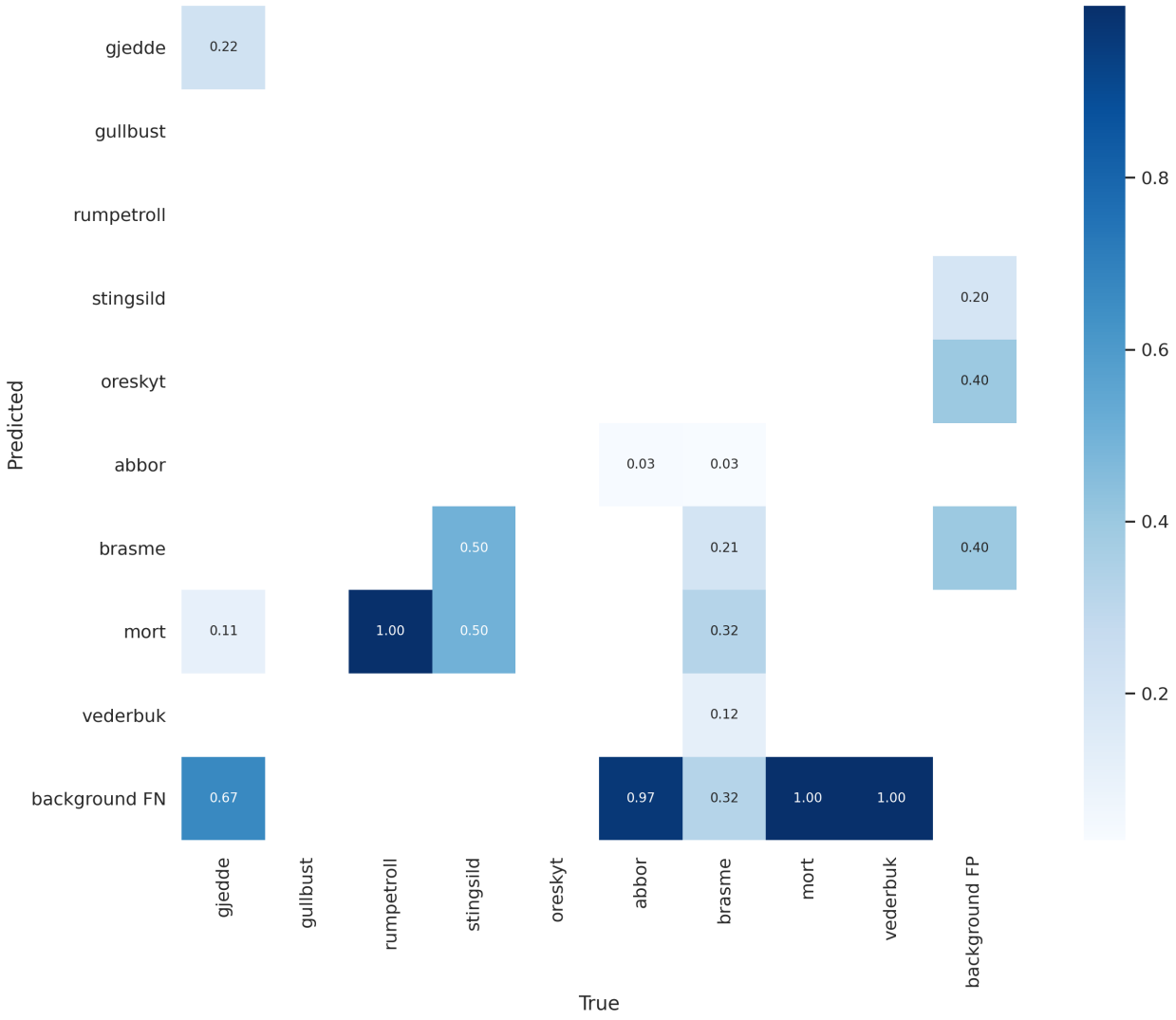


Figur 6.6: *Confusion matrix* modell fra økt nummer 1 på testdatasett.

Etter at et datasett var annotert ble mye av treningen av en modell utført som en parallel oppgave med utvikling av løsningen generelt. Dette resulterte i at det ble mindre eksperimentering med forskjellige innstillinger og modeller en det som kanskje det burde vært, for et best mulig resultat. Det ble heller prioritert å bruke tid på utvikling av kjernefunksjonalitet i løsningen.

6.4.3 Ytelse

Resultatene viser som forventet ut fra [44] at modellene som *YOLOv5m6*, hvor det benyttes en større standard bildestørrelse, bruker lengre tid en modeller som bruker mindre bildestørrelser. Alle modellene viser allikevel at de gir raskere en



Figur 6.7: *Confusion matrix* modell fra økt nummer 4 på testdatasett.

sanntid prediksjoner på en VM med GPU som definert i tabell 6.2.

6.4.4 Implementation

Denne APIen er ganske enkel og for bruk i produksjon for større prosjekt bør det vurderes å bruke en dedikert modelltjener som for eksempel *TensorRT*[8] eller *TorchServe*[9].

[8] `https://github.com/wang-xinyu/tensorrtx`, besøkt 10.05.2021
[9] `https://pytorch.org/serve/`, besøkt 10.05.2021

6.4.5 Valg av løsning

Det ble valgt tidlig i prosjektet en dyplærings tilnærming for å løse oppgaven, Dette blant annet på grunn av at dette var et fagfelt som var ønskelig å fordype seg i. Blant medlemmet som har jobbet med denne delen av oppgaven har dette gitt en større forståelse av nevrale nettverk og nettverk brukt i dyplæring. Det har også gitt en større forståelse for dybden i fagfeltet, ved at det er stor forskjell mellom å kunne benytte dyplæring metoder, og forstå konseptene, i motsetning til å utvikle nye metoder ut fra tidligere arbeid.

Ved valg av en dyplæring løsning ble kravene til datasettet større. Dette kan i etterkant vurderes, ut fra problemer som er diskutert i avsnitt 6.4.1, at en mer tradisjonelle datasyn tilnærming kunne kanskje være et bedre løsning. En datasyn løsning med bruk av et mindre datasett og en trekkbasert løsning, som diskutert i avsnitt 6.1, kunne vært et alternativ. Om dette hadde vært en bedre tilnærming er usikkert og kunne også gitt sine egne problemer. Blant annet hvor lett er det å få en god trekk-deskriptor for hver art i det varierende lys og sikt forhold som videomaterialet inneholder.

Kapittel 7

Sporing

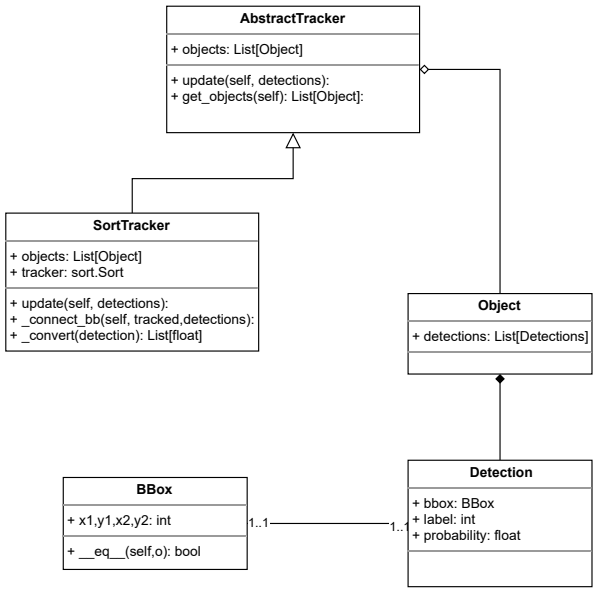
Etter at en video har blitt detektert står en igjen med avgrensningsbokser for hver deteksjon i hvert bilde. For å kunne si noe om hvor mange individuelle objekt som er detektert kan en ta i bruk objektsporing. Avsnitt 4.1.5 forteller om teorien bak objektsporing.

7.1 Implementasjon

Det ble implementert domenespesifikke klasser for objekt, deteksjon, avgrensningsboks og sporing. Disse vises i klassediagrammet i figur 7.1. BBox definerer hvordan en avgrensingboks ser ut. Klassen eksisterer for å gjøre det enkelt å sammenligne avgrensningsbokser. Detection definerer en deteksjon. Dette er kildedataen *Tracing*-pakken konverterer. Denne består blant annet av en BBox, klassifisering og sannsynlighet. Object definerer objekt, samt satt sammen av flere deteksjoner. Objektene blir resultatet av *Tracing*.

Det ble implementert en AbstractTracker, se kodeliste 7.1. Dette ble gjort for å gjøre det enkelt å lage nye spesialiseringer til forskjellige sporingalgoritmer. Abstraksjonen tilbyr et grensenitt med metodene update() og get_objects(). Funksjonen update() tar inn en liste med Detections, gjør eventuell konvertering, og sender det til den interne sporingsalgoritmen. De resulterende objektene blir plassert i self._objects.

Det ble implementert et enkelt API endepunkt for sporing, /tracking/track. Pakken er implementert som en *Offline* sporer. Dette betyr at alle deteksjoner blir sporet på ett kall. Datastrukturen som sendes inn representerer en video, der hvert element representerer et bilde. Inne i hvert av *bilde*-elementene ligger en liste med deteksjoner i videorammen. Etter sporingen er fullført blir det sendt tilbake en liste med objekter.



Figur 7.1: Klassediagram av *Tracing*-pakken med de mest essensielle medlem og metoder

```python
class AbstractTracker(abc.ABC):
    """Abstract tracker class."""

    _objects: Dict[int, Object] = dict()

    def update(self, detections) → None:
        """Update the tracker."""
        raise NotImplementedError

    def get_objects(self) → dict[int, Object]:
        """Return the object dict."""
        return self._objects

    {...}
```

Kodeliste 7.1: AbstractTracker implementasjon

7.1.1 Valg av sporingsbibliotek

For selve sporingen var det ønsket å bruke et bibliotek. Det måtte ha høy gjennomgangsrate og tilstrekkelig presisjon basert på målene fra avsnitt 1.3.2. Et annet viktig kriterie var at biblioteket måtte være enkelt å implementere og teste. *MOTChallenge*[1] ble benyttet for å få en oversikt over forskjellige sporingsalgoritmer. Her rangeres algoritmenes resultat basert på hvor godt de håndterer datasett og testparametere.

[1] `https://motchallenge.net/`, besøkt 30. april 2021

Simple Online and Realtime Tracking (SORT) [23] ble valgt basert på dens høye hastighet og tilstrekkelig resultat basert på MOT20 [51] og MOT17 [52]. SORT, som mange andre løsninger fra MOTChallenge, er et forskningsprosjekt. Det er ikke nødvendigvis laget for å bli benyttet i produksjon. Dette vises tydelig i kodebasen. Eneste kodefil fra SORT er `sort.py`[2]. Filen inneholder funksjonalitet for både kjøring som applikasjon og bibliotek. For å kjøre SORT trengtes det noen ekstra avhengigheter, for eksempel *matplotlib*, som ellers ikke behøves i løsningen.

Koden til SORT er lisensiert som GPLv3, noe som gjør at koden kan endres og publiseres av hvem som helst. For at SORT skulle fungere bedre mot gruppens løsning ble det opprettet en avgrening på ett av medlemmenes personlige GitHub konto[3]. Det ble fjernet en del kode for å konvertere SORT til et rent bibliotek. Dette fjernet også en del avhengigheter som vår implementasjon ikke trengte. Avgreningen ble også konvertert til å bli støttet av Poetry, se avsnitt 4.2.4, slik at en enkelt kunne legge det inn som en avhengighet i løsningen.

For å implementere SORT ble det laget en spesialisering av `AbstractTracker`, med navnet `SortTracker`. SORT tar i mot en avgrensningsboks og sannsynlighet som en liste, $[x_1, y_1, x_2, y_2, p]$. Resultatet fra SORT ser identisk ut med unntak at sannsynligheten p er byttet ut med en sporings-ID. Løsningens `Detection`-objekt har noe mer data enn dette, blant annet klassifisering. For å binde opp igjen et ferdig sporet objekt med sine deteksjoner ble det opprettet en funksjon, `_connect_bb()` , se kodeliste 7.2. Funksjonen sammenligner avgrensningsboksene til de urørte innsendte deteksjonene med avgrensningsboksene fra SORT. Deretter opprettes ferdige deteksjonsobjekt med nødvendig data fra de innsendte. Sammenligningen av avgrensingboksene skjer ved hjelp av `BBox.__eq__()` se, kodeliste 7.3. Ettersom avgrensningsboksene har koordinater som flyttall benyttes en toleranse på 1 %.

[2] `https://github.com/abewley/sort/blob/bce9f0d1fc8fb5f45bf7084130248561a3d42f31/sort.py`, besøkt 04. april 2021

[3] `https://github.com/R0flcopt3r/sort`

```python
class SortTracker:
    {...}
    def _connect_bb(self, tracked: np.ndarray, detect: List[Detection])
     → None:
        """Re—associate boundingboxes with a detection to determine the
     label.

        Parameters
        ----------
        tracked : np.ndarray
            Tracked objects from self.tracker
        detect : List[Detection]
            The detections to associate the boundboxes too
        """
        for t in tracked:
            t_box = BBox(*t[0:4])
            for d in detect:
                if t_box == d.bbox:
                    self._update_object(
                        int(t[4]),
                        Detection(
                            d.bbox,
                            d.label,
                            d.probability,
                            d.frame,
                            d.true_track_id,
                        ),
                    )
                    break
```

Kodeliste 7.2: Koble ramme til deteksjon

```python
class BBox:
    {...}
    def __eq__(self, o: BBox) → bool:
        """Check if the two boundingboxes are witin 1 percent of
    eachother.

        Parameters
        ----------
        o : BBox
           Other BBox

        Return
        ------
        bool :
            If they're equal
        """
        x1 = abs(self.x1 - o.x1)
        y1 = abs(self.y1 - o.y1)
        x2 = abs(self.x2 - o.x2)
        y2 = abs(self.y2 - o.y2)

        tol = 0.01
        return (
            x1 < tol * abs(o.x1)
            and x2 < tol * abs(o.x2)
            and y1 < tol * abs(o.y1)
            and y2 < tol * abs(o.y2)
        )
```

Kodeliste 7.3: Sjekk om to rammer er like

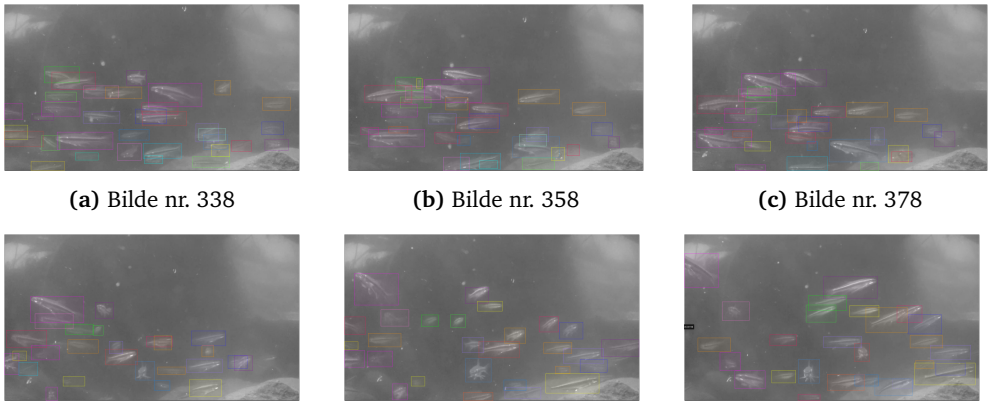
7.2 Evaluering

For evaluering benyttes *MOTA* og *MOTP* se avsnitt 4.1.5. Det ble implementert ekstra funksjonalitet i gruppens avgrening av SORT for evaluering. Falsk positiv og bom ble verifisert av forfatteren til *sort.py* [4] og commit *02b8856* [5] viser hvordan dette ble hentet ut. Hypotese og objektpar ble bestemt etter omvendt-utvilking, og hentet ut med commit *da41981* [6]. *Mismatch* er ikke hentet ut fra SORT, men kalkulert ved å sjekke om et objekt har deteksjoner med ulike true_track. Dette er et datapunkt laget for verifisering. En gren ble laget i løsningens kodebrønn. Her ble det opprettet et skript, benchmark.py se vedlegg G som tar i bruk ekstra funksjonalitet fra SORT for å gjennomføre evaluering.

Datasettet som ble benyttet under evaluering var *Ørekyt Dårlige forhold*. Dette er en av videoene oppdragsgiver hadde laget til for trening av maskinlæringsmodellen, som gruppen også hadde annotert. Figur 7.2 viser noen bilder av datasettet

[4] https://github.com/abewley/sort/issues/128
[5] https://github.com/R0flcopt3r/sort/commit/02b8856bef3972c7086b25b558e533926a274d1c
[6] https://github.com/R0flcopt3r/sort/commit/da419815de6b3d73a4280f1dd15e12b7c09a4c8a

Figur 7.2: Ørekyt dårlige forhold, skjermdump fra CVAT.



(a) Bilde nr. 338 **(b)** Bilde nr. 358 **(c)** Bilde nr. 378

(d) Bilde nr. 398 **(e)** Bilde nr. 418 **(f)** Bilde nr. 438

Totale Objekt$_{gt}$	Totale Objekt	$MOTA$	$MOTP$	$MOTP_{IoU}$
105	107	0.976	1.375	0.939

Tabell 7.1: Resultat av tracking med ørekyt stim.

fra annoteringsverktøyet CVAT. Hver farge er ett individ. Ettersom datasett er annotert av mennesker vil dette simulere absolutt beste tilfelle. Eneste variabelen vil være sporingsalgoritmen. På den måten vil en kunne ha et grunnlag for å teste sporingen. Det ble benyttet de 458 første bildene i datasettet på grunn av et kutt kort tid etter. Kuttet ville medføre mange falske positive, og invalidere forsøket.

Resultatene i tabell 7.1 viser at SORT, med dette datasettet, er innenfor våre mål bestemt fra avsnitt 1.3.2. $MOTA$ viser at vi vil ha lite feiltelling med en verdi på 0.98. Under arbeidet med pakken ble det oppdaget at det eksisterer to mål for $MOTP$, som nevnt i avsnitt 4.1.5. Tallet oppgitt i avsnitt 1.3.2 skulle vert $MOTP_{IoU}$. Målet ble satt på 0.75 for Multiple Object Tracking Precision ($MOTP$) og implementert resultat endte på 0.939. Resultatet viser at SORT klarer å forutse godt hvor fisken kommer til å svømme.

Resultatene er nok noe høyere enn det som er realistisk å se i virkeligheten. En bedre test ville vært om et menneske har manuelt sport resultat fra en deteksjonsalgoritme, for så å sammenligne mot SORT. Dette er noe gruppen ikke kunne ta seg tid til. Derfor er det blitt gjennomført med annotert data. Med annotert data er det sannsynligvis mye høyere IoU mellom avgrensningsboksene i t_n og t_{n+1} for et objekt.

7.3 Videre arbeid

Denne pakken fikk minst jevnlig oppmerksomhet gjennom prosjektet. En fungerende prototype ble raskt implementert. Etter dette ble arbeidskraft plassert mer mot kjernefunksjonaliteten. Litt finpuss ble gjennomført mot slutten av utviklingsperioden, men her finnes det fortsatt en del forbedringer.

Ved mer tid skulle det blitt utført forsøk på flere datasett. *Ørekyt Dårlige forhold* inneholder stort sett fisk som ikke overlapper og svømmer uniformt. Det er derimot svært mange og vil kanskje derfor fungere best som en kort ytelsetest. Det kunne også vært fordelaktig å hoppe over noen bilder på tilfeldige intervall for å forsøke å produsere et mer rotete datasett.

I videre arbeid burde det gjøres mer testing av sporingspakken for å verifisere at resultatene stemmer. Implementering av nye sporingalgoritmer burde være relativt greit, men for å enklere evaluere de trengs det mer arbeid. En måte for å unngå modifisering av brukte biblioteker kunne vært å benytte rammeverk som TrackEval[7]. Det burde også sees nærmere på `SortTracker._connect_bb()`. 1 % toleranse ved sammenligning av avgrensningsbokser førte til tilfeller under testing der flere deteksjoner ble ansett som den korrekte. Dette betyr at det er høyere sannsynlighet for *mismatch*. En mulig løsning kan være å sammenligne dens sporings-ID med forrige deteksjon. Et annet alternativ kan være å endre SORT slik at den kunne ta imot inn en ekstra ID parameter. På den måten ville oppkobling av bokser bli lettere i etterkant, uten å medføre feil.

Bruk av ett enkelt application programming interface (API) endepunkt kan vise seg i fremtiden å være problematisk. Vi har ingen kontroll over hvor mange potensielle deteksjoner som kan finnes i en lang jobb. Det kan derfor være mulig at API-et ikke klarer å handtere datamengden. Videre arbeid ville vært å dele opp mengden data i mindre arbeidsmengder. En mulig løsning kan være å introdusere noen ekstra endepunkter for å gjøre sporingen *Online*. Eksempelvis kan sporingen ha et endepunkt til å opprette en *tracker* med ID for kjørende jobb. Et annet endepunkt vil oppdatere *tracker* fortløpende med nylig sporet data. Til sist vil et annet endepunkt hente ut sporede og delvis sporet objekter.

[7] `https://github.com/JonathonLuiten/TrackEval`

Kapittel 8

Kjernefunksjonalitet

Selve kjernefunksjonaliteten i applikasjonen er å prosessere videoer og føre statistikk. Til å utføre dette trengs det en kjerne som syr sammen hele applikasjonen, og oppbevarer data. Resultatet av dette er Python-pakken core. I dette kapittelet vil en beskrive hvordan modulen ble implementert, samt noen problemer som oppstod under utvikling.

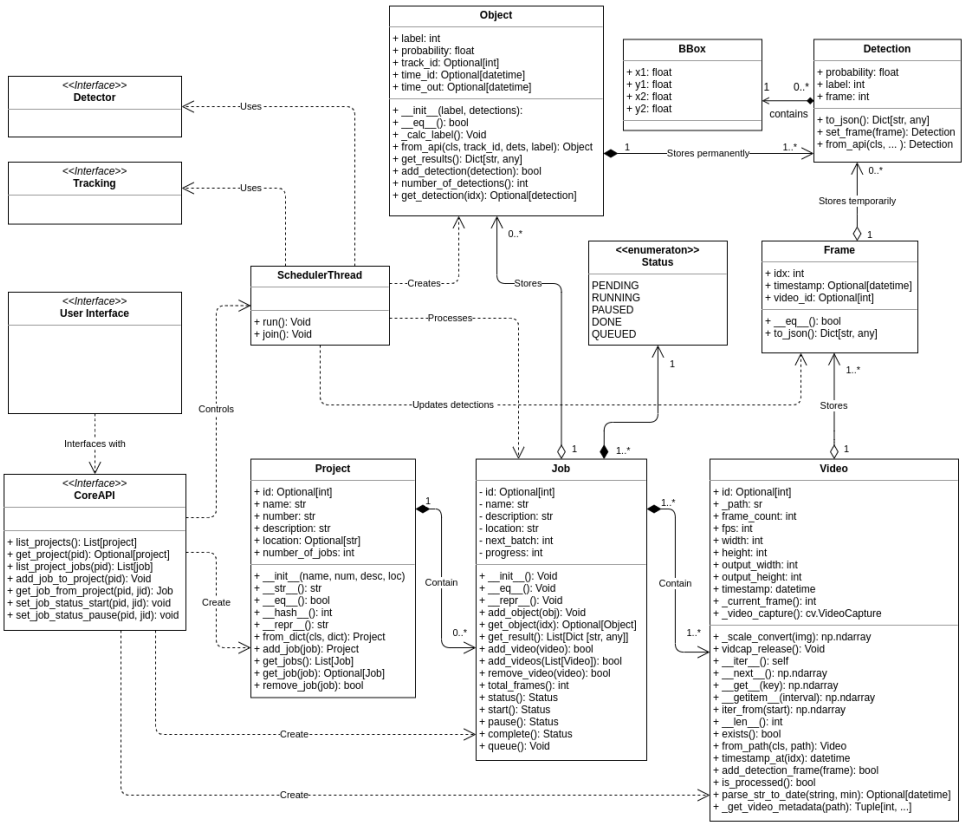
8.1 Modulens oppbygging

Ut fra planlagt domenemodell ble det utviklet et konsept for hvordan core modulen skal fungere. Se figur figur 8.1 for et oppdatert klassediagram over modulen. Brukergrensesnittet bruker core sitt API til å utføre handlinger. Blant annet å opprette prosjekt, jobber og videoer. Grensesnittet vil også kommunisere med scheduler tråden for å starte en jobb. Mer om hvordan denne klassen fungerer i avsnitt 8.3.

8.2 Lagring av data

Core modulen bruker *repository pattern* fra DDD til å lagre unna objekter til disk. Det ble laget tre spesifikke oppbevaringssted ("repository") til grupperte objekter. Hvert oppbevaringsted har en abstrakt klasse som har rolle som grensesnitt for en type objekt. Dette gir fleksibilitet for å implementere andre måter å lagre unna objekter på. Til bakenden ble det valgt å bruke biblioteket *SQLAlchemy*, se avsnitt 4.2.8.

Det første oppbevaringsstedet ProjectRepository håndterer prosjekt og jobber. Prosjekt kan ha mange jobber, derfor ble de gruppert under prosjekter. Jobber inneholder også Video-objekter som skal prosesseres. Flere videoer kan potensielt inngå i forskjellige jobber. Dermed kan de tilhøre mange ulike prosjekter. Video -objekter kan også være nyttig å hente ut uavhengig av en jobb. For eksempel funksjonalitet med å kunne se bilde av en deteksjon. Det vil også være lettere å implementere sjekk om video allerede er prosessert når Video-objekt har sitt eget



Figur 8.1: Klassediagram av core modul.

oppbevaringsted. Siden en slipper å sjekke gjennom alle jobber, i alle prosjekter. Derfor ble det laget et eget oppbevaringsted VideoRepository.

Under prosesseringen vil objekter bli opprettet som et sluttresultat. Derfor ble det laget et ObjectRepository som lagret unna Object-klasser. For at sporingen skal skape objekter bruker den Detection-objekter som et Object er basert på. Siden Object aldri vil eksistere uten minst en Detection ga det mening at oppbevaringstedet for Object inkluderte underliggende Detection- objekter.

Ved implementasjonen av fremgangslagring under prosessering ble det tydelig behov for å lagre unna enkelt Detection-objekter. Siden prosesseringen syr sammen en eller flere Detection-objekter til Object-objekter, vil det ikke gå å lagre unna deteksjoner med ObjectRepository. Problemet oppstod sent under utviklingen av løsningen, og det var ikke attraktivt å gjøre store endringer på datastrukturen. Derfor ble Video-klassen modifisert til å inneholde en liste med Detection-objekter. Dette gav prosesseringen mulighet å lagre unna deteksjoner før de var sydd sammen til fullverdige objekter. Til ettertanke ville det være bedre å splitte deteksjon fra objekt i ObjectRepository, for å lage et DetectionRepository. Dette vil ha gjort det lettere å lagre unna deteksjoner uten å måtte knytte de spesifikt mot videoer.

Valget med å bruke *SQLAlchemy*, se avsnitt 4.2.8, til implementasjon av oppbevaringstedene hadde noen fordeler. Det sparte tid på implementasjon av lagring og skriving til fil, samt konvertering til eksempelvis JSON. For å holde modellen til core ren for spesifikk implementasjon av *SQLAlchemy* ble det valgt å bruke *classical-mapping*[1]. Noe som betydde at en trengte en fil som spesifiserer datafelt og relasjoner til hvert objekt som skal lagres. Ulempen med dette er at om en skal legge til vedvarende variabler i et slikt objekt, må en modifisere to filer. Først i klassen som endres, for så å oppdatere *SQLAlchemy* sin kartlegging. Dette opprettholder DDD sitt prinsipp om *dependency inversion* siden infrastrukturen under er avhengig av implementasjonen i domenet, og ikke omvendt. Siden en må modifisere to filer, kan det oppstå forvirring om noen skal utvide løsningen senere. På grunn av manglende tid og kompleksitet i løsningen, ble dette ikke prioritert å løse.

8.3 Køhåndtering

For å kunne sette flere jobber i kø fra brukergrensesnittet var det behov for å laget et køsystem. Resultatet av dette var klassen SchedulerThread. Denne fungerer som en wrapper rundt den innebygde thread-klassen i Python og dens oppgave er å fange programfeil som oppstår. Hovedpoenget med *scheduler* er å vedlikeholde en trådsikker kø job_queue. API vil kunne sende prosjekt- og jobb-ID til køen, for så at scheduler-tråden henter ut de. Deretter vil scheduler kalle prosesseringfunksjonen.

Slik som scheduler er implementert er det denne tråden som faktisk prosesse-

[1] `https://docs.sqlalchemy.org/en/13/orm/mapping_styles.html#classical-mappings`, besøkt 16.05.2021

rer en jobb. Planen under utvikling var at scheduler laget en ny prosess som utførte prosessering på en jobb. Scheduler skulle bare håndtere hvilke jobber som kjøres, køen med jobber, og kontrollere kjørende jobb med prosesssignaler. Slik som det er implementert nå vil ikke scheduler-tråden kunne hente ut nye meldinger fra køen under prosessering. Først etter en jobb er prosessert vil scheduler hente ut neste melding fra job_queue. Dette betyr at funksjonalitet som å stoppe eller sette en jobb på pause er vanskelig å implementere. API har også et endepunkt for å pause, men på grunn av manglende tid ble ikke implementasjonen prioritert.

Kontroll av kjørende prosess blir gjort av innebygd Event klasse i Python. Scheduler har et event som den bruker til å sjekke om det er på tide å stoppe. Om eventet ikke er sett lengre, vil scheduler og kjørende prosessering stoppes. Dette er fordi eventet fra scheduler blir passert som argument til prosesseringsfunksjonen. Når eventet blir deaktivert, for eksempel under feil i prosesseringen, vil også scheduler stoppe. Det beste hadde vært å ha et eget stopp-event for selve prosesseringen. På den måten vil scheduler kunne forsette å starte jobber om kjørende prosessering får en feilmelding. Bruker vil heller ikke få informasjon om at scheduler har stoppet i brukergrensesnittet. Noe som kan skape forvirrelse siden nye jobber ikke vil starte før programmet er startet på nytt. Dette er noe som var ønsket å forbedre, men gruppen hadde ikke nok tid til dette.

Under utvikling viste det seg at en ikke kan opprette nye tråder eller prosesser under et API-kall til *FastAPI*. En ønsker heller ikke at API skal vente med å svare klienten til jobben er prosessert. Derfor ble det nødvendig å sette opp et kommunikasjonssystem mellom tråder ved hjelp av en kø.

Etter stopp av utvikling, ble det klart at mye kunne gjøres annerledes med prosesskommunikasjon. Eksempelvis kunne biblioteket Celery[2] brukes for kommunikasjon og oppgave håndtering (*task*). FastAPI har også integrasjon mot biblioteket, noe som ville gjort kommunikasjonen mer ryddig. Celery kunne også blitt brukt på selve prosesseringen, som vil ha gjort det lettere siden en kan dele inn deteksjon og sporing i to oppgaver. Ved hjelp av Celery sin oppgavesyntaks vil en kunne garantere at sporingen alltid skjedde etter alle videoene er detektert. Celery introduserer også noe mer kompleksitet. Til eksempel må en ha en *broker* for at prosessene skal kommunisere.

8.4 Lasting av video

Under utvikling ble det klart at det trengtes en metode for å dele opp videoer inn i mindre arbeidsmengder. Lastes en hel video inn i maskinens minne blir det raskt tom for minne. For å løse dette ble det implementert et abstraksjonslag for videoer for å redusere mengden påkrevd minne.

Videoformat som *mp4* eller *mkv* inneholder effektive komprimeringsalgoritmer som gjør at filene blir mindre, men på bekostning av tapt informasjon [53]. For at deteksjonen skal kunne prosessere videorammene er det nødvendig at de

[2] `https://docs.celeryproject.org/en/stable/index.html`, besøkt 16.05.2021

er i et NumPy-format. Noe som forårsaker at bildedata ikke kan lengre være komprimert [54].

Hver piksel i NumPy-arrayet blir representert med tre *uint8* integer verdier. Størrelsen per piksel tilsvarer 3 x 1 Byte, en byte per RGB fargekanal. Om en multipliserer dette med oppløsningen deteksjonen bruker på 640x360, tilsvarer dette 691,2 kB minne for en videoramme. NINA har videoer på opptil 30 minutter på 25 bilder per sekund, som tilsvarer 45000 rammer. Den totale mengden minne en ville trenge for å holde en video i minne blir 31,1 GB per video. Dersom en velger å bruke andre opptrente modeller på høyere oppløsninger vil minneforbruket bli mangedoblet.

Måten minneforbruket ble løst på var å introdusere en ny hjelpeklasse *VideoLoader*, som har i oppgave å abstrahere bort lastingen av videoer. Klassen produserer en liste med videorammer opptil en maksimal batch- størrelse. Hvor alle assosierte data og videorammer blir midlertidig returnert med en *yield*[3]. Dette løser problemet med minneforbruket da en video kan deles opp i mindre batcher. Bruk av *yield* i klassens generatorfunksjon tillater at videofilen blir holdt åpen i OpenCV, og programmet kan raskt hente ut neste ramme uten å reinitialisere.

Siden VideoLoader-klassen skaper et abstraksjonslag på alle videoer blir det lettere å implementere selve prosesseringslogikken. Klassen vil opprette batcher sømløst over flere videofiler. Prosesseringen trenger ikke holde orden på hvilken fil den jobber på, all lasting og indeksering blir utført av *VideoLoader*. Lagring av fremgang er også trivielt å implementere på grunn av at hver batch kan representeres som en arbeidsenhet i *unit of work pattern*[55]. Klassen gjør også indeks- og kontekstspesifikk informasjon tilgjengelig for hver batch. For eksempel tidsstempel til en gitt ramme, rammeindeks over alle videoer og nåværende batchnummer. På denne måten slipper en komplekse kalkulasjoner for å hente ut indekser eller filstier i en batch.

8.5 Prosessering av videoer

Tidlig i planleggingsfasen til prosesseringen ble det klart at det kunne oppstå feiltellinger ut fra når sporingen kjører. Om sporingen kjører etter hver video vil den potensielt telle fisk to ganger om fisken er i bildet mellom overgangen av to videoer. For å løse dette ble det diskutert to potensielle løsninger. Det mest åpenbare ville være å modifisere sporingen til å håndtere flere videoer. Det ble vurdert som tidkrevende og usikkert på grunn av dårlig dokumentasjon av sporingsbiblioteket og at sporing var nytt blant gruppa. En annen løsning var å gjøre prosesseringen todelt. Først detekter fisk i alle videoer, for så å spore alle deteksjonene til slutt. På grunn av vanskeligheter med modifisering av sporing ble det valgt å gjøre prosesseringen i to steg.

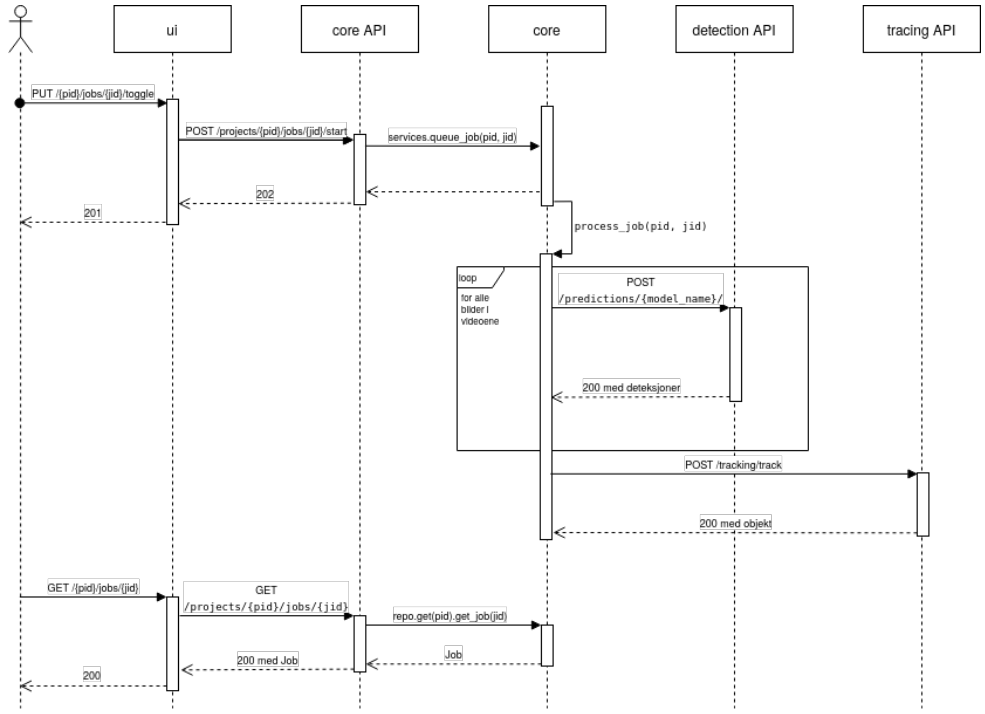
Siden oppdragsgiver ønsker å kjøre løsningen på en arbeidstasjon, var det

[3] `https://docs.python.org/3/reference/simple_stmts.html#the-yield-statement`, besøkt 16.05.2021

viktig at fremgang lagres underveis. Prosesseringen tar ofte veldig lang tid med NINA sine videofiler. Derfor ville det være urealistisk å forvente at en hel jobb blir ferdig før programmet eller maskinen stoppes. Det som tar mest tid er å utføre er deteksjon av videoene.

For å lagre fremgang ble det implementert en variabel i Job-klassen for å holde orden på hvilken batch fra VideoLoader som var utført. Dette gjorde til at VideoLoader kan gjenoppta fra jobbens sist prosesserte batch. For å lagre unna deteksjonene ble Video klassen modifisert til å inneholde Frame-objekter, som igjen inneholder Deteksjon- objekter. Når en påbegynt jobb gjenopptas vil den hente ut deteksjoner fra videoene i jobben. På den måten vil prosesseringen gjenopptas dersom programmet avsluttes.

prosesseringen av en jobb er den avhengig av kommunikasjon til deteksjon- og sporingsgrensesnitt. Denne kommunikasjon er illustrert med sekvensdiagrammet i figur 8.2 fra en bruker starter en jobb i brukergrensesnittet.



Figur 8.2: Sekvensdiagram for kommunikasjon mellom modulene ved start av en jobb fra brukergrensesnitt.

Om en bruker starter en jobb i brukergrensesnittet vil jobben bli lagt i kø for prosessering igjennom et kall til core-grensesnittet. Når det er ledige ressurser vil prosesseringen av jobben starte. Den vil da gå over alle bildene i videoene til jobben. Underveis vil den sende en liste med noen bilder av gangen til deteksjonsgrensesnittet, til alle videoene har blitt detektert. Deretter sendes listen over alle bildene med tilhørende deteksjoner til sporingsgrensesnittet.

Funksjonaliteten her er svært kompleks. Mye av koden handler om å holde

orden på er hva som skal skje dersom programmet avsluttes før en jobb er ferdig. Tilstanden må lagres og jobben må settes på pause. En del av denne kompleksiteten kunne flyttes til en funksjon som kjørte når programmet startet. Denne funksjonen ville tilbakestilt alle jobbene som var markert som kjørende til pauset. Dette ville vert en mye mer robust måte, da programmet kan oppleve en fatal feil og allikevel kunne restarte jobbene som var igang før det skjedde.

8.6 Bestemme tidspunkt for objekt

Et av oppdragsgiver sine ønsker var å vite når et objekt ble synlig på skjermen og når det forsvant. Filnavnet til videofilene levert av oppdragsgiver besto av et generisk navn etterfulgt av tidsstempel og løpenummer. Se kodeliste 8.1 for eksempel. En *video* består av flere videoklipp på 30 minutt. Hver videofil bygger 30 minutter på den neste.

```
File1 –[2020–07–18_08–17–54]–000.mp4
File1 –[2020–07–18_08–17–54]–001.mp4
File1 –[2020–07–18_08–17–54]–002.mp4
File1 –[2020–07–18_08–17–54]–003.mp4
```

Kodeliste 8.1: Eksempel filnavn for videofil

Funksjonen parse_str_to_date() ble implementert. Denne benytter et *Regular Expression*-uttrykk[4], se kodeliste 8.2, for å hente ut en streng med tidsstempel og eventuelt løpenummer om det eksisterer. Strengen blir så konvertert til et datetime-objekt og hvor hvert 30·løpenummer minutter blir addert på samme objekt. Resulterende datetime-objekt returneres som det endelige tidsstempelet.

```
"\[\d{4}(–\d{2}){2}_(\d{2}–){2}\d{2}\](–\d{3})?"
```

Kodeliste 8.2: Regular Expression for tidsstempel og løpenummer

Når en jobb blir prosessert vil hvert Frame-objekt motta et tidsstempel. Når objekt blir returnert fra sporing hentes ut bildenummeret til alle deteksjonene som bygger opp objektet. Disse blir så sortert, og første og siste nummer blir brukt som indeks i listen over Frame og dens tidsstempel blir hentet ut, se kodeliste 8.3.

[4] https://en.wikipedia.org/w/index.php?title=Regular_expression&oldid=1022839132p, besøkt 18.05.2021

```
for o in objects:
    times = sorted([det.frame for det in o._detections])

    time_in = frames[times[0]].timestamp
    if time_in == None:
        raise RuntimeError("Expected type datetime, got None")
    o.time_in = time_in

    time_out = frames[times[-1]].timestamp
    if time_out == None:
        raise RuntimeError("Expected type datetime, got None")
    o.time_out = time_out
```

Kodeliste 8.3: Algoritme for å assosiere tidsstempel for inn og ut på objekt

Denne funksjonaliteten la et krav på at videofilene måtte inneholde tidsstempel i filnavnet. Tidsstempelet må også følge nøyaktig den standarden som oppdragsgiver har benyttet. Ideelt burde det vært umulig å hente ut dette fra videofilenes metadata, da dette kunne hentes nærmest direkte. Når gruppen kopierte videofilene over til et annet mer praktisk lagringmedium ble denne dataen nullstilt. Det ble derfor bestemt å heller benytte tidstempel i videofilene ettersom dette er en standard som oppdragsgiver har fulgt på samtlige av videofilene de leverte.

8.7 Optimalisering

Optimalisering handler om å finne områder i koden som ikke fungerer optimalt. Dette omhandler som oftest om ytelsesproblematikk. Under kommer avsnitt 8.7.1 og 8.7.2. Disse følger IMRAD-formatet, og er begge selvstendige. Her avdekkes og løses problematikk rundt ytelse i *core*-pakken.

8.7.1 Uthenting av bilder frå video

Løsningen leser ut et gitt antall bilder fra en video for å sende disse til deteksjon. En video på 30 minutt med en bildefrekvens på 25 Hz resulterer i 45000 bilder. Det er derfor viktig for vår løsning at denne prosessen går fortest mulig. Et vanlig og velkjent verktøy for å operere på video er FFmpeg[5]. Basert på gruppens erfaring med verktøyet benyttet løsningen originalt et Python-bibliotek for FFmpeg[6]. Under testing var det oppdaget at løsningen tok svært lang tid. Opp mot 3 ganger lengre enn sanntid. Det var på grunn av dette gjort undersøkelser for å bestemme hvor tiden ble brukt. Ved hjelp av profilering ble det oppdaget at mesteparten av tiden gikk på å klargjøre data før det ble sent til deteksjon, mer spesifikt FFmpeg. Uthentingen tok mellom 10 sekunder til 5 minutter ut fra hvor langt den var kommet i en 30 minutters video.

[5] `https://ffmpeg.org/`, besøkt 27.04.2021

[6] `https://github.com/kkroening/ffmpeg-python`, besøkt 29.05.2021

CPU	Intel i7 8700k (6c/12t@4.3GHz)
RAM	64GB (3200MHz/CL16)
Disk	NVME PCIe SSD (3200/1800 MB/s)
OS	Fedora 33

Tabell 8.1: Spesifikasjon for PC 2

Metode	Total	Gjennomsnitt	Median	Standardavvik
OpenCV	3.042	0.006	0.006	0.001
FFmpeg	239.388	0.479	0.479	0.19

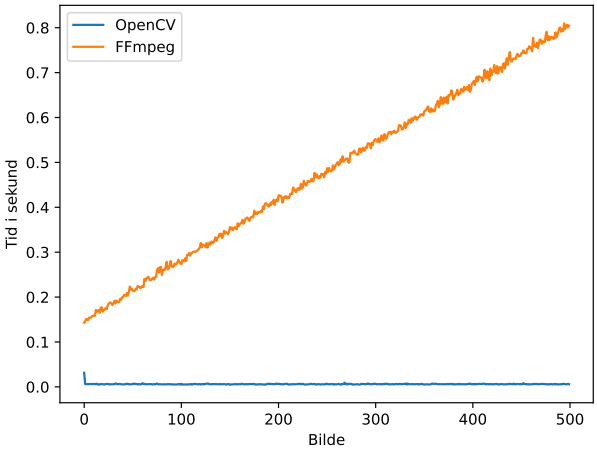
Tabell 8.2: Resultat av OpenCV v. FFmpeg med 500 bilder. Kjørt på PC 2 som vist i tabell 8.1

For testing ble det produsert et Python-skript, vedlegg J. Her ble det importert to varianter av core.model.Video klassen med implementasjon av OpenCV og FFmpeg. Maskinen som ble benyttet i forsøkene vises i tabell 8.1. Det ble ikke gjort tiltak for å renske prosessorens mellomlagring mellom forsøkene.

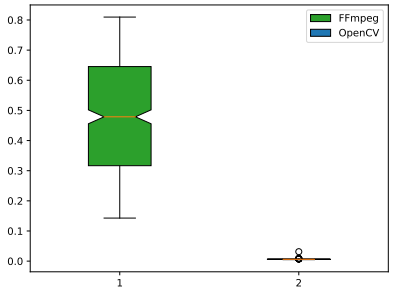
Det ble gjennomført tester med FFmpeg og OpenCV, se vedlegg K og L for de respektive implementasjonene av __iter__, __next__ og __get_item__. Umiddelbare resultat fra tabell 8.2 viser 79 ganger høyere totaltid med FFmpeg kontra OpenCV. Gjennomsnitt og median viser tid for hvert bilde. Resultatet viser derimot ikke at vår implementasjon med FFmpeg bruker lengre tid desto lengre ute i videoen den henter bilde fra. Figur 8.3a viser FFmpeg sitt økende tidsbruk over tid, kontra OpenCV sin mer eller mindre konstante tidsbruk per bilde. Argumentet for å benytte OpenCV forsterkes i figur 8.3b. Her vises det tydelig hvor treg FFmpeg er. Figur 8.3c viser OpenCV for seg selv slik at det er mulig å se dens målinger.

For å uthente bilder med OpenCV, må en instansere et cv2.VideoCapture-objekt[7]. Implementasjonen i vedlegg L har plassert dette i __iter__(). Dette gjør til at vi kan bare kalle __next__, som vil benytte den allerede eksisterende VideoCapture instansen. Denne vil kalle self._video_capture.read(), som henter ut bildet i nåværende punkt, for så å inkrementere enn intern teller i VideoCapture. Dette er i motsetning til FFmpeg-implementasjonen der en egen teller må inkrementeres, og en ekstra funksjon __get_item__() må kalles. OpenCV sitt VideoCapture-objektet må lukkes etter bruk. Under prosesseringen må dette gjøres etter hver fullført video med Video.vidcap_release(). Dette er derimot ikke et problem når en benytter __get_item__() eller __get__() i videoklassen. Funksjonene har mulighet til å lukke VideoCapture instansen selv. Det blir også antatt at Python sin søppelinnsamling er tilstrekkelig for å slette VideoCapture instansen dersom en skulle glemme å gjøre deg selv.

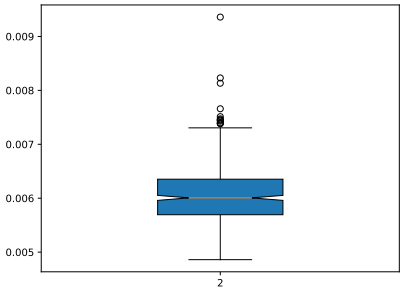
[7] `https://docs.opencv.org/4.5.2/d8/dfe/classcv_1_1VideoCapture.html`, besøkt 29.04.2021



(a) Lineplot av OpenCV vs. FFmpeg med 500 bilder



(b) Boxplot av OpenCV vs. FFmpeg med 500 bilder



(c) Lineplot av OpenCV vs. FFmpeg med 500 bilder

Figur 8.3: OpenCV vs. FFmpeg ved lasting av video

```
def bytesio(img):
    byte_io = io.BytesIO()
    Image.fromarray(img).save(byte_io, "png")
    byte_io.seek(0)
    return byte_io
```

Kodeliste 8.4: bilde til byte implementasjon med BytesIO

```
def opencv(img):
    img = cv.cvtColor(img, cv.COLOR_BGR2RGB)
    img_byte = cv.imencode(".png", img)
    return io.BytesIO(img_byte)
```

Kodeliste 8.5: bilde til byte implementasjon med OpenCV

Basert på resultatene og de minimale problemene med Video.vidcap_release() ble FFmpeg erstattet med OpenCV.

8.7.2 Konvertering av bilde til byte

Når bilder skal sendes over til deteksjons API var det nødvendig å oversette bildedata fra numpy.ndarray til bytes kodet som bilder. Original implementasjon var ved bruk av io.BytesIO, se kodesnutt 8.4. Ved profilering ble det oppdaget at denne prosessen tok omlag 6 sekunder per *batch*[8] med 50 bilder.

For testing ble det produsert et lite Python skript, som benytter VideoLoader[9], se vedlegg I. Både BytesIO og OpenCV ble testet. Begge bibliotekene ble også testet med multiprocessing.Pool[10]. For å fordele arbeidet mest mulig likt over prosessorkjerner var chunk sett til batch_size//os.cpu_count(). Dette deler listen med bilder opp slik at hver kjerne har omtrent like mye data å jobbe med[11]. Maskinvare brukt vises i tabell 8.3. Ingen tiltak ble utført for å renske prosessorens mellomlagring mellom forsøkene.

Resultat vises i tabell 8.4, og viser en tydelig forskjell på OpenCV og BytesIO implementasjonene. OpenCV på en prosessorkjerne er fortsatt raskere enn BytesIO på 12 kjerner. Derfor ble ikke multiprosessering brukt i løsningen. Til tross for at multiprocessing biblioteket til Python skal være *threadsafe*, oppstod det uforklarlige feil som resulterte i at core kresjet. Det er mulig at dette skjedde da scheduler kjører som en tråd under core, og at når en tråd oppretter prosesser så gikk noe galt.

[8] En *unit of work*, liste med bilder.

[9] Klasse for å produsere ein batch.

[10] `https://docs.python.org/3/library/multiprocessing.html#module-multiprocessing.pool`, besøkt 29.04.2021

[11] `https://docs.python.org/3/library/multiprocessing.html#multiprocessing.pool.Pool.map`, besøkt 29.04.2021

CPU	Intel i7 8700k (6c/12t@4.3GHz)
RAM	16GB (3200MHz/CL16)
GPU	nVidia GTX 1080 (Driver v. 460)
Disk	SSD SATA 6Gbps
OS	Ubuntu 20.10

Tabell 8.3: Spesfikasjon for PC 1

Metode	En kjerne	Tolv kjerner
OpenCV	633s	388s
BytesIO	3559s	764s

Tabell 8.4: Tid i sekund brukt for konvertering av `numpy.ndarray` til byte med PC beskrive i tabell 8.3. Rundet til nærmeste hele sekund.

Dette er spekulering og ikke verifisert. Å rette dette ville uansett tatt for lang tid til at gruppen kunne implementert dette. Forskjellen med multiprosessering for OpenCV-implementasjonen viser heller ikke like god skalering som BytesIO. BytesIO er $4x$ tregere uten multiprosessering, OpenCV er knapt $2x$, og dette over 12 kjerner.

8.8 Diskusjon

Det ble jevnt over fokusert lite på feilhandtering. Noe som førte til at core-pakken ofte kræsjet når en prøvde å bruke den. Det ble mye arbeid mot slutten for å ordne feilhandtering rundt programmets terminering og det finnes fortsatt tilfeller der programmet blir terminert på akkurat rett tidspunkt til at det kræsjer. Feilmeldinger ble heller ikke alltid sendt tilbake til API-et, slik at de som benytter funksjonaliteten fikk beskje om at noe gikk galt. Dette er noe som burde vært inne i planleggingsfasen, bestemmelser rundt hvordan feilmeldinger skulle finne veien tilbake til API. Noe av årsaken til manglene her kan være gruppens erfaring med utvikling av kommandolinjeverktøy. Med slike program er det godt nok at feil blir printet ut i terminal og programmet termineres.

Ytelsen er svært god. Det finnes et område der det kan forbedres ytterligere. Når bilder sendes for deteksjon kunne kjernen begynt å forbrede neste sett med bilder. Når kjernen da får svar fra deteksjon vil da det neste bildene i beste fall være klare, og sendes umiddelbart. Resten av pakken fungerer nært optimalt. Det spekuleres at for betre ytelse så må en over på kompilerte metoder, eksempelvis ved bruk av teknikker som Just-in-Time kompilering[12].

[12] `https://en.wikipedia.org/w/index.php?title=Just-in-time_compilation&oldid=1019401428`, besøkt 19.05.2021

Kapittel 9

Brukergrensesnitt

9.1 Introduksjon

Fra oppgaven var det ønskelig med et brukervennlig og visuelt grensesnitt. Tidlig i utviklingsfasen ble det vurdert som enten en lokal eller en webløsning. Å utvikle brukergrensesnittet som en webløsning ville være med på å fylle dette kravet. Ved at man lager det som en webløsning betyr at alle kan bruke løsningen i sin egen nettleser. En kan til og med kjøre brukergrensesnittet på en annen maskin i nettverket. Det er heller ikke nødvendig å installere det lokalt på maskinen. I tillegg er det enkelt å utvikle en webløsning ved hjelp av teknologiene beskrivet i kapittel 4 da spesifikt HTML og CSS. Det ble allerede til første møte med oppdragsgiver laget en demo av hvordan webløsningen kunne se ut. Dette reflekterte hvor enkelt det ville være å utvikle brukergrensesnittet som en webløsning. På grunn av disse fordelene, ble en webløsning valgt.

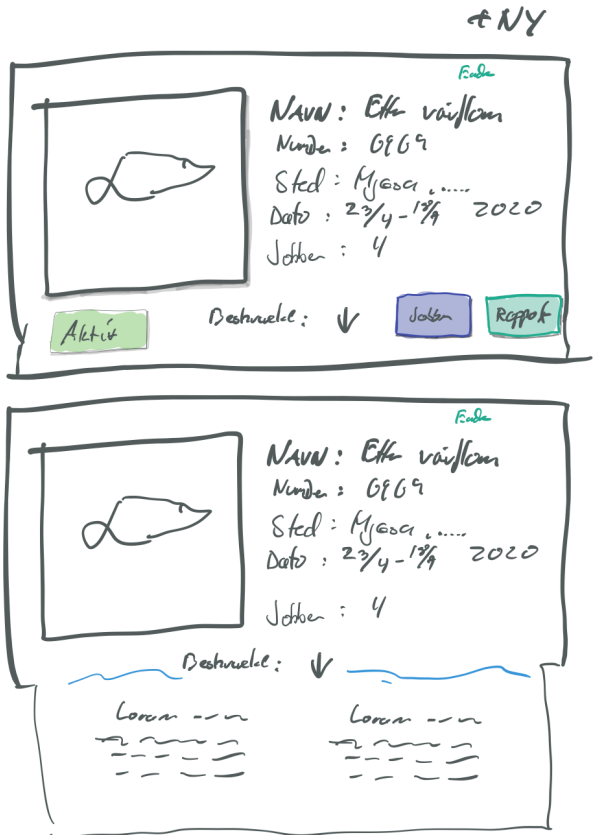
9.1.1 Skisser

Underveis ble skisser laget for hvordan gruppen tenkte brukergrensesnittet kunne se ut. Disse representerer ikke hvordan det endelige produktet endte opp. Skissene var nyttige til å gi oppdragsgiver en enkel representasjon før utvikling startet. Under utvikling ble det gjort endringer, men essensen er hentet ut fra skissene.

Skissen i figur 9.1 viser en prosjektboks. Denne ble skissert tidlig med tanke på at mye informasjon skulle vises i en boks. Figur 9.2 viser en jobbside. På toppen er litt generell statistikk og på bunnen er resultatstabell.

9.2 Metode

Et rammeverk brukes for å gjøre det enklere å utvikle webløsninger. Rammeverk tilbyr ofte funksjonalitet for å generere html i sanntid. Dette betyr at webløsningene kan virke mer dynamiske for brukeren, men i virkeligheten er siden statisk. Valget av rammeverk hadde som kriterier at det var enkelt, innholdt bare det løsningen trengte, lett å lære, raskt å sette opp og var skrevet i Python.



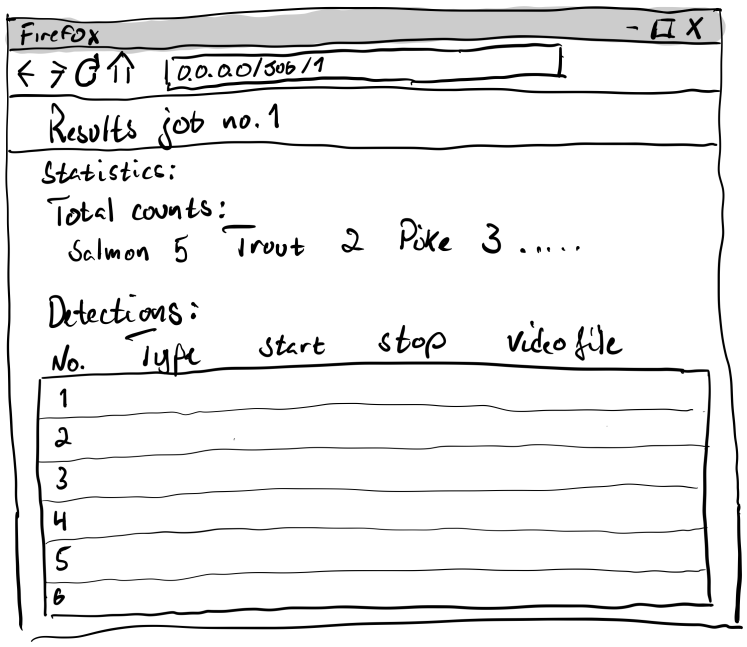
Figur 9.1: Skisse av prosjektboks

Django er et at de største webrammeverkene for Python. Det er listet opp som nummer én i den offisielle wiki-siden til Python[1]. Det blir referert *The Most Popular Python Web Frameworks in 2021* [56] til som et full-stack rammeverk. Dette betyr at det inkluderer det meste som trengs for å sette opp en webløsning pluss mer. Dette er blant annet en innebygd ORM, automatisk generering av administrasjonsside, søkefunksjon og sending av e-post. Deretter så gruppen på de offisielle kom-i-gang-veiledningen. Django starter med å instruere brukeren om å lage en spesifikk filstruktur [57]. Hvor da hver fil/mappe hadde sin egen funksjon og tilhørende kode.

Flask er et mindre rammeverk som også nemnes på Python sin wiki-side. Flask er et mikrorammeverk, som betyr at det er kun det mest nødvendige som er med av funksjoner. Med utvidelser kan ekstra funksjonalitet bli tilført rammeverket. Flask sin kom-i-gang-veiledning instruerer brukeren til å legge all koden i en og samme fil [58]. Dette betyr ikke at rammeverket praktiserer at alt kode skal ligge i en fil. Men at det fra start ga litt innblikk i hvor omfattende rammeverket virket. Gruppen var ikke klar for å ta i bruk et som krevde mye for å komme i gang.

En tredje faktor var artikkelen *Comparative study on Python web frameworks:*

[1] `https://wiki.python.org/moin/WebFrameworks`, besøkt 18.05.2021



Figur 9.2: Skisse av en jobb m/ resultat

Flask and Django [59] som bekreftet flere av de tidligere punktene. Basert på denne og det øvrige punktene , valgte gruppen å få for Flask. Les mer om valget i diskusjonen avsnitt 9.4.

9.2.1 Flask

Etter Flask ble valgt, startet prosessen med å lære rammeverkets funksjonalitet. Siden gruppen ikke hadde stor kjennskap til Flask fra før, ble mye tid i starten brukt til å lese den offisielle dokumentasjon[2]. Dokumentasjonen ble flittig brukt gjennom utvikling. En YouTube-spilleliste av Corey Schafer[3] ble brukt for å lære seg konseptene til rammeverket. Samt ble Stackoverflow[4] brukt til bl.a. å finne svar på feilkoder. Innebygde konsepter som ble brukt fra Flask var *blueprints*, *application factory* og *error handling*.

Under følger en mer utdypende forklaring av de forskjellige funksjonene som ble benyttet fra Flask.

Application Factory pattern

Ved bruk av application factory pattern kan det gjennom den innebygde funksjonen create_app() lages en instans av applikasjonen. Det har som fordel at funksjonen

[2] `https://flask.palletsprojects.com/en/1.1.x/#user-s-guide`, besøkt 11.05.2021

[3] `https://www.youtube.com/playlist?list=PL-osiE80TeTs4UjLw5MM6OjgkjFeUxCYH` besøkt 04.05.2021

[4] `https://stackoverflow.com/questions/tagged/flask`, besøkt 04.05.2021

returnerer en instans med medsendte parametere. Disse parameterene kan sendes med som en liste, eller sti til en fil som inneholder disse, se kodeliste 9.1 som eksempel. En annen fordel er at funksjonen vil foreta feilsjekker før du mottar instansen. Dette vil forsikre at de satte parametere er gyldige.

```
def create_app(test_config=None):
    app = Flask(__name__)

    if test_config is None:
        app.config.from_pyfile("config.cfg", silent=True)
    else:
        app.config.from_mapping(test_config)
```

Kodeliste 9.1: Application factory

Ved hjelp av parameteren test_config kan applikasjonen benytte forskjellige parametere i ulike miljøer. Under testing kan det være aktuelt å bruke en annen database, endre endepunktet for API-kall og slå på mer utfyllende logging.

Utviklingsmønsteret er ofte brukt i andre sammenhenger og programmeringsspråk. Da ikke nødvendigvis som en *application factory*, men mer generelt med en fabrikk funksjon som skaper objektet med tilsendte parametere. Dette fungerer på samme måte som over. For eksempel når en ny klient til et API-grensesnitt skal opprettes, kan man sende med en URL-adresse som forteller hvor klienten skal kjøre kallet mot. Her er målet å sjekke at adressen er gyldig.

I vår løsning ble patternet brukt for å enkelt lage instansen hvor den skulle kjøre. Blant annet når hele løsningen skulle startes ble det brukt et oppstartskript. Dermed var det enkelt å instansiere applikasjonen ved hjelp av funksjonen over. I tillegg var det tenkt å brukes oppstartskript til testing, men som avsnitt 9.4 kommer innpå, ble ikke dette tatt i bruk.

Blueprints

I Flask benyttes blueprint[5] for å dele opp applikasjonen i biter som omhandler unike funksjoner. Som for eksempel kan det være en administrasjonsside som brukeren kan nå ved å navigere seg til `http://example.com/administrasjon`. Alt som har med denne siden kan da ligge som et egen *blueprint*.

Essensen går ut på at man tar kode som hører til en funksjon eller endepunkt og flytter det i sin egen mappe. Hvert endepunkt i webløsningen har sitt eget *blueprint*. Fordelene med dette er at man har kun ett sted å jobbe på den koden som er tilhørende uten å røre annen kode. Separasjonen gjør det også mulig å benytte samme *blueprint* i andre Flask-applikasjoner. Siden en kan flytte pakken mellom de, eller bruke samme *blueprint* flere ganger under ulike endepunkt.

I starten av utviklingen var alt av brukergrensesnitt samlet i samme fil. Med ny kunnskap om *blueprint* gikk gruppen inn for å separere mest mulig av bruker-

[5] `https://flask.palletsprojects.com/en/1.1.x/tutorial/views/`, besøkt 04.05.2021

grensesnitt koden. Tanken var å gjøre webløsningen mer modulær og oversiktlig. Det var også for å kunne legge til flere moduler og videre utvikling av ny funksjonalitet. Dette var blant annet en administrasjonsside for å kunne justere på parameterene i løsningen. En slik funksjonalitet ble ikke med. Se avsnitt 9.4 for tanker rundt dette.

```python
import os

from flask import Flask, render_template

from ui.projects.projects import construct_projects_bp


def create_app():
    app = Flask(__name__)

    projects_bp = construct_projects_bp(app.config)
    app.register_blueprint(projects_bp, url_prefix="/projects")

    @app.route("/")
    def index():
        return render_template("index.html")

    @app.errorhandler(404)
    def page_not_found(e):
        return render_template("404.html"), 404

    return app
```

Kodeliste 9.2: Instansiering av Flask

```python
def construct_projects_bp(cfg: Config):
    """Create constructor from function to pass in config."""
    projects_bp = Blueprint(
        "projects_bp",
        __name__,
        template_folder="templates",
        static_folder="static",
    )
```

Kodeliste 9.3: Konstruksjon av en Flask blueprint

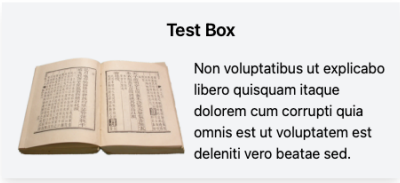
Error handling

Error handling med Flask går ut på å definere en funksjon som håndterer HTTP feilkoder. For eksempel kan dekoratoren `@app.errorhandler(404)` legges på før definisjonen til funksjon som skal håndtere HTTP 404 feilkode. Dette resulterer i at når brukeren får feilen, er det denne funksjonen som blir aktivert og kan vise en egen side som forklarer feilen. Se kodeliste 9.2 hvor siden `404.html` returneres i

webløsningen. Feilkode 404 oppstår oftest når tjeneren ikke finner den ressursen brukeren prøver å navigere seg til. Det finnes mange forskjellige HTTP feilkoder[6], men webløsningen endte opp med håndtere noen få av disse.

9.2.2 HTML/CSS

Utviklingen av HTML/CSS foregikk mest med prøv-og-feil metoden. Se avsnitt 9.4 diskusjon, for hvordan dette muligens ikke var beste måte å jobbe på. For å utføre prototyping raskt og enkelt, ble rammeverket Tailwind[7] brukt. Ett av gruppemedlemmene hadde erfaring fra før med Tailwind. Siden det var raskere å se endringer i HTML og CSS med rammeverket, ble det bestemt at det skulle brukes. Det ble også brukt som en del av det endelige brukergrensesnittet.



Figur 9.3: Eksempel på bruk av CSS ©Wiki-Pedia

Det meste av stilene ble skrevet direkte i HTML ved hjelp av Tailwind' klasser. Noe som senere ble flyttet ut i en egen fil. For eksempel ble stilene for knapper tatt ut da mye av det var likt. Knappene har kun noen få endringer basert på de forskjellige funksjonalitetene. Derfor ville det være enklere å kun måtte skrive <button class="btn btn−start"> for å fortelle at knappen skal ha den generelle `btn`-klassen og den spesialiserte `btn-sart`-klassen for "start jobb".

CSS sammenlignet med Tailwind

For å illustrere Tailwind, er det inkludert et lite eksempel. Det tar utgangspunkt i å gjenskape figur 9.3 ved å gi koden i kodeliste 9.4 samme stil på forskjellige måter. Hvor kodeliste 9.5 viser CSS-koden som legges i egen fil eller hvor kodeliste 9.6 viser det samme, men ved bruk av Tailwind.

[6] `https://developer.mozilla.org/en-US/docs/Web/HTTP/Status`, besøkt 19.05.2021
[7] `https://tailwindcss.com`, besøkt 04.05.2021

```
<div>
  <h2>Test Box</h2>
  <img
    src="https://upload.wikimedia.org/book.png"
    alt="An open book."
  />
  <p>
    Non voluptatibus ut explicabo libero quisquam itaque dolorem cum
    corrupti quia omnis est ut voluptatem est deleniti vero beatae sed.
  </p>
</div>
```

Kodeliste 9.4: Ren HTML før CSS-klasser

```
div {
  background-color: #f3f4f6;
  margin: 0 auto;
  padding: 1rem;
  width: 28rem;
  box-shadow: 0 0 #0000, 0 0 #0000, 0 10px 15px -3px rgba(0, 0, 0, 0.1)
    ,
    0 4px 6px -2px rgba(0, 0, 0, 0.05);
}

h2 {
  display: block;
  font-size: 1.25rem;
  font-weight: 600;
  margin-bottom: 1rem;
  text-align: center;
}

img {
  float: left;
  margin-right: 0.5rem;
  width: 12rem;
}
```

Kodeliste 9.5: Vanlig CSS

```
<div class="p-1 mx-auto w-112 bg-gray-100 shadow-lg">
  <h2 class="block mb-1 font-semibold text-xl text-center">Test Box</h2
    >
  <img
    src="https://upload.wikimedia.org/book.png"
    class="float-left mr-2 w-48"
    alt="An open book."
  />
  <p>
    Non voluptatibus ut explicabo libero quisquam itaque dolorem cum
    corrupti quia omnis est ut voluptatem est deleniti vero beatae sed.
  </p>
</div>
```

Kodeliste 9.6: Tailwind m/ utiliy-klasser

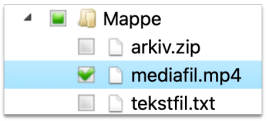
9.2.3 JavaScript

For å gjøre brukergrensesnittet mer interaktivt, ble JavaScript benyttet for å manipulere HTML-elementene. JavaScript kan endre Document Object Model (DOM). Denne representerer den statiske HTML strukturen til en nettside. Her vil JavaScript kunne trekke ut et element som et objekt, som eksempelvis <p> nevnt i forrige delkapittel. Dermed kan en utføre JavaScript, som `el.classList.add(btn-green)` når brukeren gjør en handling. I dette eksemplet legges det til en ny klasse. Hvor "el" er elementet <p>, classList er nåværende liste og `add(btn-green)` legger til en ny klasse på listen.

Eksempel på dette er når brukeren starter en jobb, da endres knappen med en gang for å vise endringen. Her benyttes det som kalles Asynchronous JavaScript and XML (Ajax) som gjør det mulig å sende informasjon asynkront. Uten JavaScript er det nødvendig å poste til endepunktet (en fastsatt adresse som svarer på forespørselen), for så å laste siden inn på nytt. Med JavaScript kan kallet kjøres asynkront i bakgrunnen. Ved hjelp av å manipulere DOM, kan knappen endres uten at brukeren opplever at siden lastes inn på nytt.

Filbehandler i JavaScript

Siden løsningen skal prosessere videofiler, var det nødvendig med av et system som lot brukeren kunne velge filer fra et filtre. Det var også viktig at bruker kan velge filer som ikke nødvendigvis er på samme maskin som klienten. Vanlig HTML filvelger støtter ikke å velge filer fra andre filsystem enn det tilgjengelig på kilentens maskin. Derfor var det behov for et filvelger rammeverk, og jsTree[8] ble benyttet. Det ble valgt fordi det var gratis å bruke, som flere av alternativene ikke var. Biblioteket var også lite og inneholdt



Figur 9.4: Eksempel på et filtree i jsTree.

[8] `https://www.jstree.com`, besøkt 11.05.2021

kun den funksjonalitet som var ønsket. Se figur 9.4 for hvordan et sånt tre ser ut, og se kodeliste 9.7 for hvordan man bygger strukturen til jsTree. JavaScript Object Notation (JSON) blir brukt for å representere strukturen filtreet skal vise. Fordelen med at JSON brukes er at det er enkelt å generere strukturen i bakenden uten å tilføre mye ekstra kode. På den måten kan en lage en filvelger som tillater bruker å velge filer i en trestruktur, uavhengig av hvor i systemet den er plassert.

```
{
  "id": "mappe",
  "text": "Mappe",
  "icon": "ikon_mappe.png",
  "state": {
    "opened": true
  },
  "children": [
    {
      "id": "fil_arkiv",
      "text": "arkiv.zip",
      "icon": "ikon_fil.png"
    },
    {
      "id": "fil_media",
      "text": "mediafil.mp4",
      "icon": "ikon_fil.png",
      "state": {
        "selected": true
      }
    },
    {
      "id": "fil_tekst",
      "text": "tekstfil.txt",
      "icon": "ikon_fil.png"
    }
  ]
}
```

Kodeliste 9.7: Rådata for jsTree

9.2.4 Optimalisering av CSS og JavaScript

Før løsningen settes ut i produksjon, benyttes et postprosesserings-bibliotek for å redusere størrelse på koden. Denne vil optimalisere koden for utrulling. Bakgrunnen for dette er at under utviklingen kan for eksempel CSS-filene være spredt etter funksjon. Dette gjøres ofte for å lettere finne frem hvor du trenger å endre. Om alt lå på samme sted, ville man fort ende opp med veldig lange filer som ville være slitsomme å navigere. Med egne filer har alt sin logiske plass. Ved hjelp av optimalisering vil en ha god ytelse, kombinert med at det er enkelt å gjøre endringer.

Eksempel på denne optimaliseringprosessen kan en se om vi tar koden i kodeliste 9.5 og kjører denne gjennom CSSNANO[9]. Dette er et verktøy for å komprimere CSS. Den produksjonsklare koden inneholder nå ingen mellomrom eller tomrom. Fordelen er at filen som må lastes ned er mindre når brukeren først åpner siden. Det samme vil også gjelde når lignende prosess kjøres på JavaScript-filene.

```
div{background-color:#f3f4f6;margin:0 auto;padding:1rem;width:28rem;box
    -shadow:0 0 #0000,0 0 #0000,0 10px 15px -3px rgba(0,0,0,0.1),0 4px
    6px -2px rgba(0,0,0,0.05)}h2{display:block;font-size:1.25rem;font-
    weight:600;margin-bottom:1rem;text-align:center}img{float:left;
    margin-right:0.5rem;width:12rem}
```

Kodeliste 9.8: Komprimert CSS-kode

Med JavaScript vil det i tillegg til komprimering foregå enda en prosess. Fordi for eksempel jsTree er en egen pakke, brukes et eksternt verktøy[10] for å laste ned pakken. Dette oppsettet fungerer ikke i produksjon da filene ligger på forskjellige steder. Med postprosessering kan man ved hjelp av @import jstree; i vår egen fil fortelle programmet at i produksjonsfilen skal jsTree være inkludert. Om denne prosessen ikke hadde blitt gjort, ville det nå være nødvendig å inkludere tre filer; jQuery, jsTree, og vår egen fil. Fordelen her er at i stedet for å laste inn alle filene hver for seg, trengs nå kun å lastes ned én fil.

9.3 Resultat

Videre fortelles det litt om hvordan den endelige løsningen ble. Først starter det med et flytskjema av navigeringen i figur 9.5 og deretter gås det gjennom skjermdumper av de ulike sidene.

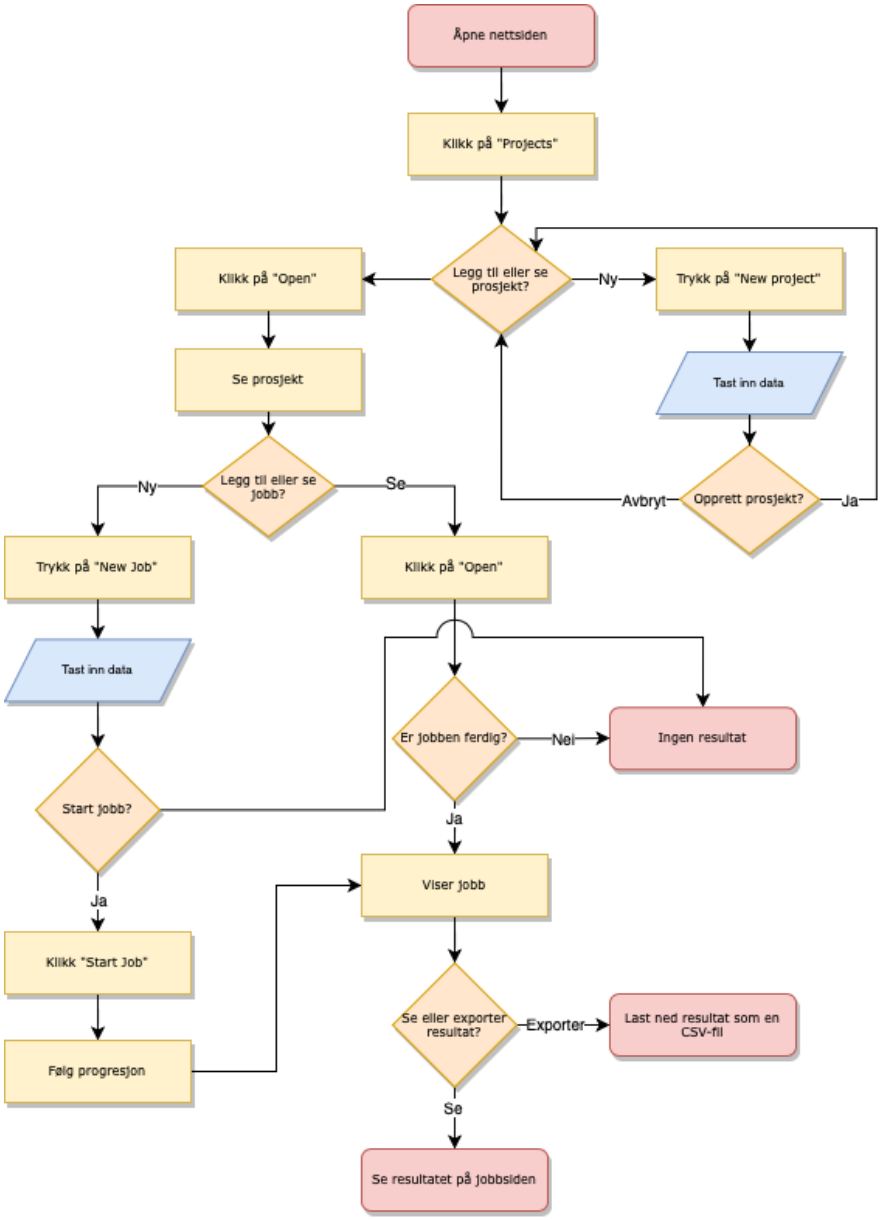
9.3.1 Flytskjema

Figur 9.5 viser en komplett oversikt over hvordan flyten i programmet er. Som figuren viser er det lagt opp slik at all interaksjon forgår gjennom brukergrensesnittet. Det starter med å åpne nettsiden, fra der navigeres det frem til et resultat.

9.3.2 Endelig design

Etter mange utviklingiterasjoner er det mye brukergrensesnitt å vise. Denne seksjonen vil vise en del figurer fra det ferdige brukergrensesnittet i løsningen. Figurene prøver å følge flyten som vil ville være naturlig å bevege seg gjennom løsningen.

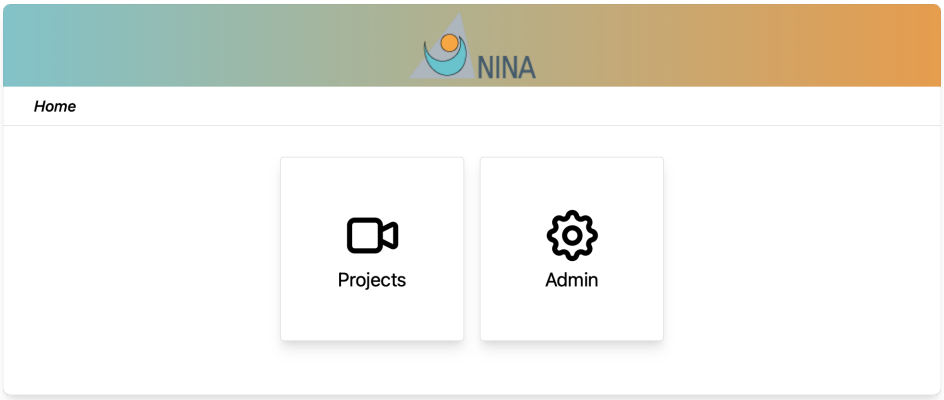
[9] `https://cssnano.co/`, besøkt 13.05.2021
[10] `https://www.npmjs.com`, besøkt 13.05.2021



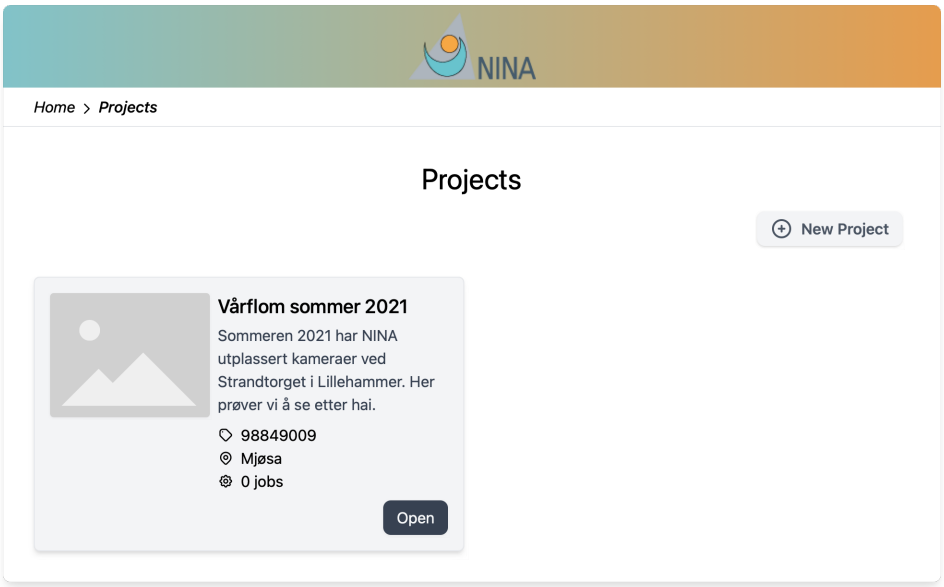
Figur 9.5: Flytskjema for brukergrensesnitt

Figur 9.6 er starten når brukeren åpner nettsiden for å ta i bruk løsningen. Her vises to store knapper, henholdsvis “Projects” og “Admin”. “Projects” er det som er implementert i løsningen. “Admin” ble det ikke tid til å implementere. Disse tilsvarer egne blueprints som forklart i avsnitt 9.2.1.

Figur 9.7 er prosjektvisningen. Hvert prosjekt er representert av en boks, som inneholder det viktigeste for å raskt identifisere prosjekter. Det endelige resultatet prøvde å forestille figur 9.1. Sammenlignet med skissen ble det endelige resultatet



Figur 9.6: Brukergrensesnitt: Startside

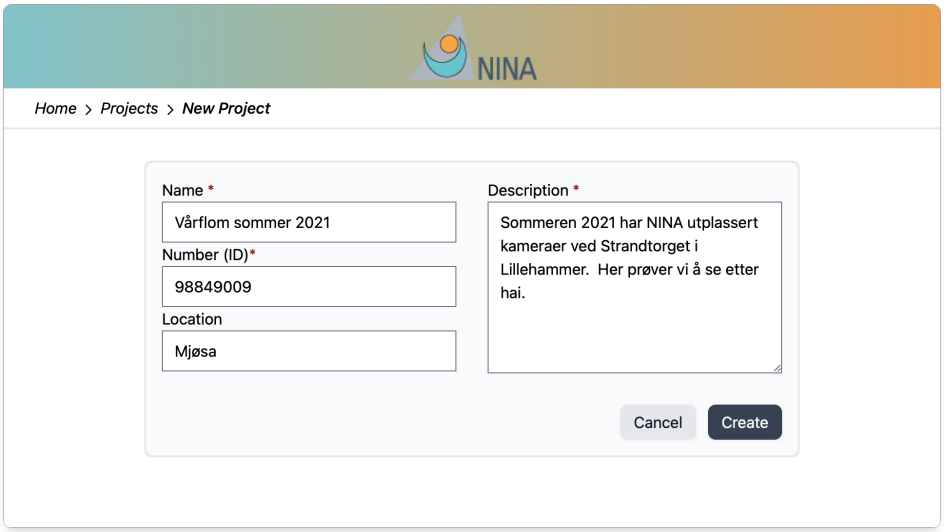


Figur 9.7: Brukergrensesnitt: Oversikt av prosjekter

mer beskjedent. Under utviklingen ble det klart at ikke alt var like nødvendig. Det ligger inne for at hvert prosjekt kan ha et bilde, men det ble ikke implementert.

Rett under logoen ligger en sti (breadcrumb). Dette er tenkt til å benyttes som en måte å navigere seg tilbake. Samtidig viser den også hvor du er i løsningen. Basert på hvor dypt du er kommet, kan man alltid navigere seg tilbake ved å trykke på et tidligere steg. Denne ble implementert på grunn av observasjonen gjort under brukertesten forklart i avsnitt 10.3.3.

Når et nytt prosjekt skal lages, kan brukeren trykke på “New Project” knappen på prosjektoversikten. Dermed vises dialog for å opprette et nytt prosjekt,



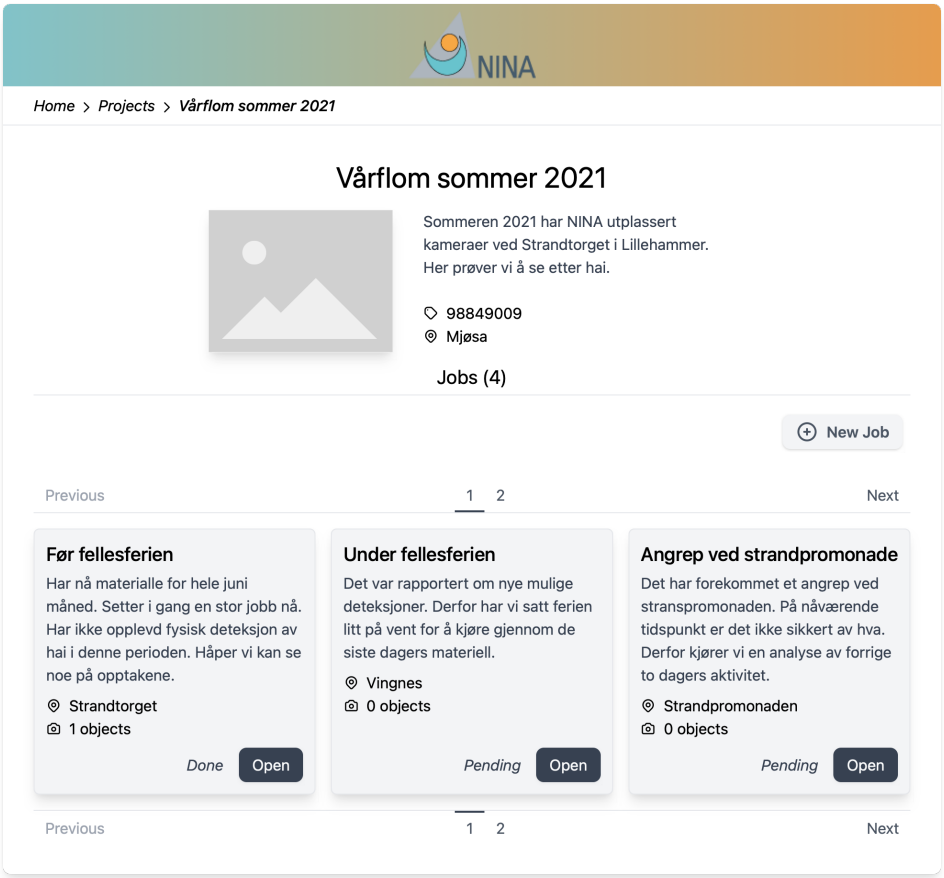
Figur 9.8: Brukergrensesnitt: Nytt prosjekt

se figur 9.8. Feltet for "Number (ID)" er et felt bruker kan gi en prosjektkoden til prosjektet. Dette er ikke en ID som blir brukt videre i av systemet, men bare som et identifikasjonverktøy til brukerne. Feltet "lokasjon" endte opp med å være statisk, og tar bare en tekststreng for plassering. Tanken var at dette skulle være en nedtrykksmeny der en kan velge mellom tidligere plasseringer, men dette ble det ikke tid til. Det ville også vært aktuelt å legge "ny plassering" under administrasjonsiden hvis den ble implementert.

Inne på et prosjekt, som figur 9.9 viser, ser en all tilhørende informasjon om et prosjekt. Dette er samme informasjon som i prosjektoversikten, men nå vises mer detaljert prosjektinformasjon. Under informasjonen ligger alle jobbene tilhørende prosjektet. Jobblistingen følger det samme stil som prosjektoversikten. Om et prosjekt har mange jobber vil brukergrensesnittet automatisk dele jobbene over flere sider. Standard vises 12 jobber, som samsvarer med tre kolonner og fire rader. Ved hjelp av medsendte parametre kan brukeren velge hvor mange det skal vises på hver side. Det er i dag ikke konfigurerbart, men kan utvides senere.

På samme måte som det opprettes et ny prosjekt, trykker bruker på "New Job" og ender opp som vist på figur 9.10. Her er feltene arrangert litt annerledes, og i tillegg vises det nå en filvelger. Filvelgeren kan leses mer om i avsnitt 9.2.3. Bruker kan velge hvilke velge mediefiler som utgjør jobben. Her er det lagt opp til at brukeren må velge riktig filer, siden det ikke går an å endre på en i etterkant. Dersom bruker ikke fyller inn felt riktig eller velger filer som ikke er støttet, vil det vises en feilmelding. Bruker kan dermed prøve på nytt med riktig data eller avbryte opprettingen. Ved feil vil den inntastede teksten komme opp igjen slik at brukeren ikke trenger å fylle disse inn på nytt.

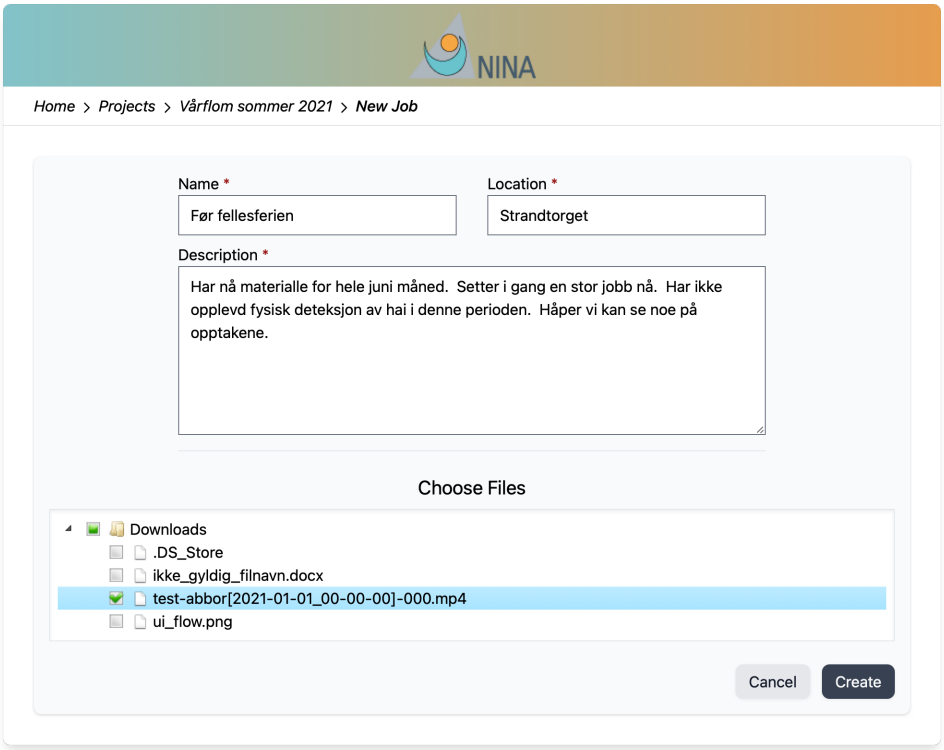
Figur 9.11 viser oversikt over en jobb. Siden jobben ikke er prosessert, vises



Figur 9.9: Brukergrensesnitt: Prosjektside

ingen statistikk. For å starte jobben benyttes "Start Job"-knappen. Det er denne knappen som er omtalt i avsnitt 9.2.3 og endrer seg basert på status på jobben. Knappen viser i tillegg et prosenttall for å vise progresjon. Se figur 9.12 hvordan denne forandrer seg. Det er også en fane som lister opp videoer jobben har.

Det siste steget på reisen er når jobben endelig er prosessert. Se figur 9.13 for jobbside med resultat. Nå er startknapp og progresjonstatus byttet ut med liste over resulterende objekter. Her vises også de kolonnene som oppdragsgiver mente var nødvending, for eksempel "Video ID" kolonnen. Ved hjelp av videolisten kan oppdragsgiver manuelt verifisere at oppdaget fisk stemmer overens med videofilene. Det er også mulig å eksportere resultatet ved å trykke på knappen "Export to CSV" oppe i høyre hjørne for resultatstabellen.



Figur 9.10: Brukergrensesnitt: Ny jobb

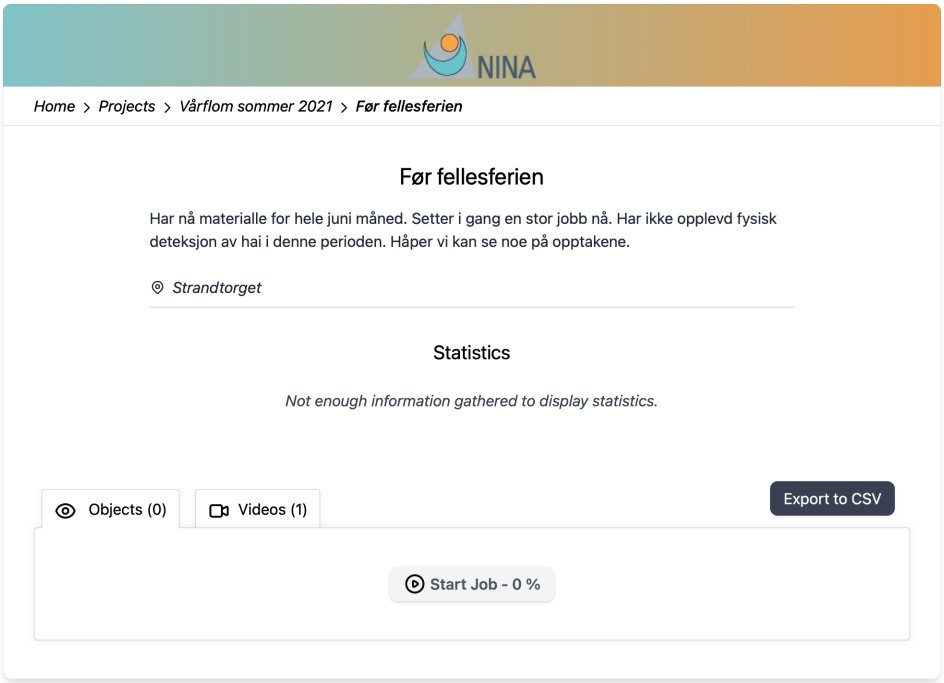
9.4 Diskusjon

Etter mange iterasjoner, var oppdragsgiver godt fornøyd med brukergrensesnittet. Resultatet av brukertesten (vedlegg F, ref. spm. #13) bekrefter dette. Oppdragsgiver var med fra første dag på design av brukergrensesnittet. Samtidig som det ble utviklet i iterativ prosess. Dette førte til at oppdragsgiver fikk de et brukergrensesnitt som passet nøyaktig sine behov.

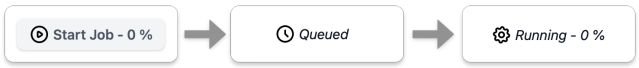
9.4.1 Implementering

Implementering av brukergrensesnittet som en webapplikasjon er noe gruppen mener var riktig. For implementering ble det benyttet Python som alle gruppemedlem var kjente med. Dette gjorde det mulig for alle å bidra med det bakenforliggende. Derimot var det bare ett medlem som hadde tidligere erfaring med generell webutvikling. Valg som Tailwind bidro til å gjøre det vanskeligere for andre medlemmer å bidra, basert på manglende erfaring.

Selve valget om å benytte seg av Tailwind kan også ses tilbake på som en fordel. Under prototypingen var det veldig greit å kunne bare endre koden i én fil for øyeblikkelig endring. Selv om vanlig CSS fungerer bra, gir klassene til Tailwind mye hjelp. Det at klassene er forhåndsbestemt, tar hånd om mye ekstra som ellers



Figur 9.11: Brukergrensesnitt: Jobb ikke begynt



Figur 9.12: Brukergrensesnitt: Progresjonsknapp

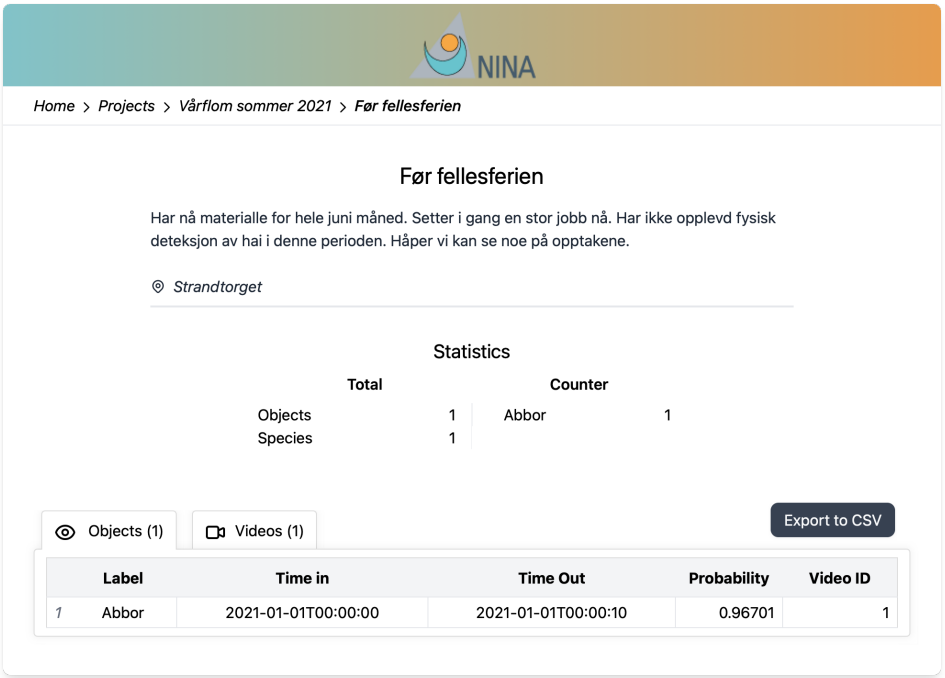
gruppen måtte implementere.

Valget om bruk av Flask kontra Django mener gruppen var det riktige valget. Sluttproduktet benytter grunnleggende funksjonalitet fra Flask. Derfor ville bruken av Django medføre at enda flere funksjoner stod ubrukt. At Django trengte mer investeringer til å komme i gang, kan også ha gjort det vanskeligere for gruppemedlem å forstå webløsningen. Flask er i tillegg et populært rammerverk som gjorde det mulig å raskt finne svar om det oppstod problemer.

Et alternativ til Flask er FastAPI sin webtjener, Starlette. FastAPI ble benyttet i de andre pakkene, og derfor kunne det være naturlig å bruke det i brukergrensesnittet også. Ettersom grensesnittet ble prototypet med Flask før prosjektstart i Januar, hadde ikke gruppen kjennskap til FastAPI. Det ble heller ikke sett på som produktivt å skrive om prototypen til Starlette.

9.4.2 Kritikk

Brukergrensesnittet ble hovedsaklig utviklet av ett gruppemedlem. Dette var et problem fordi mange valg ble tatt på egenhånd. Dette kunne være valg av ele-



Figur 9.13: Brukergrensesnitt: Ferdig jobb m/ resultat

menter og plasseringer av disse. Samt verktøyene som beskrevet i avsnitt 9.2.4. Resultatet av dette endte opp med at de andre utviklerene fikk større og større avstand til grensesnittet. Hvorvidt den endelige webløsningen kom dårlige ut som konsekvens av dette, er det vanskelig å måle. Siden oppdragsgiver var fornøyd, kan det anses at det var godt utført. Det er uansett noe gruppen burde ha løst under utviklingen. For medlemmet som jobbet med webløsningen ble det mye å gjøre, og lite tilbakemeldinger fra resten av gruppen. Et mulig tiltak vil kunne være parprogrammering, slik at flere av medlemmene er med på ta valg. Samtidig vil partner også få erfaring med hvordan webløsningen fungerer. En ville også passet på hverandre, moderert utviklingen og passet på at tid ble kun brukt på det mest nødvendige.

Oppdragsvier gav mye frihet rundt hvordan brukergrensesnittet skulle se ut. Oppgaveteksten inneholdt ingen direkte krav til brukergrensesnittet. Dette førte til at mye tid ble brukt til å utvikle skissene til prototyper. Konsekvensen av valget om en smidig utviklingsmodell gjorde også planleggingen mer flytende og ikke satt i stein tidlig. Det ville nok vært mer fordelaktig at hele gruppen jobbet sammen i starten for å lage skissene. Disse kunne blitt videreutviklet ved hver iterasjon ettersom ny funksjonalitet i bakenden var implementert.

Den største kritikken er at hverken tester eller universell utforming ble implementert i brukergrensesnittet. Dette ble rett og slett ikke priortert av utvikler. Noe av grunnen til at universell utforming ikke ble prioritert er at prosjektets bru-

kergruppe ikke var avhengig av å få det implementert. For testing ble *application factory pattern* beskrevet i avsnitt 9.2.1 tatt i bruk for å gjøre dette enklere. Den utvidede funksjonaliteten for testing ble ikke tatt i bruk. Sammenlignet med testene i de andre modulene, var testing av brukergrensesnittet avhenging av å imitere bakenden. Det ble gjort litt undersøkelse rundt dette, men det så ut til å være krevende. Dermed var det ikke dedikert tid til testing av brukergrensesnittet.

9.4.3 Manglende funksjonalitet

Under avsnitt 9.2.1 nevnes det en administrasjonsside. Denne ble tenkt på som en måte å konfigurere løsningen gjennom brukergrensesnittet. Et ønske fra oppdragsgiver var muligheten til å endre verdier for hvor nøyaktig deteksjonen skulle være. I tillegg som nevnt i avsnitt 9.3.2 er det lagt opp til skrive inn en plassering på prosjekter og jobber. Første tanken var at det skulle lages som en egen tabell i databasen. Dermed ville plasseringsfeltet fungert som en nedtrykksmeny i brukgrensesnittet. Disse skulle da være mulig å endres gjennom administrasjonssiden. Det var ønskelig å legge til nevnt funksjonalitet, men på grunn av størrelsen av resten av systemet var det for lite tid.

9.4.4 Konklusjon

Gruppen ble gitt oppgaven å lage et brukergrensesnitt for et system for artsgjenkjenning av fisk i undervannsvideoer. Som et krav skulle dette være brukervennlig og enkelt. Dette er noe gruppen opplever den endelige webløsningen oppfyller. Det samme har oppdragsgiver uttrykt gjennom møter og brukertest. Selv om gruppen på veien har møtt på utfordringer, har ikke disse vært alvorlige nok til å påvirke sluttresultatet. Slik det står i dag, oppfyller det kravene og fungerer bra opp mot bakenden av systemet.

Ved å lansere løsningen som åpen kildekode håper gruppen at den manglende funksjonaliteten kan bli tatt hånd om. Dette kan være av gruppen selv eller av andre som kommer etter. Fokuset på hva som burde prioriteres videre i webgrensesnittet er feilsjekking, tester og universell utforming. Dette vil være bra for videre bruk siden det vil forbedre bruker- og utvikleropplevelsen.

Kapittel 10

Kvalitetssikring

Diverse system ble tatt i bruk for å sørge for at kvaliteten på koden var god og ble ivaretatt. Systemene er delt inn i tre grupper; kodekvalitet, samarbeid(review) og testing. Første er for å sikre at alle i gruppen opprettholder lik kvalitet og kodestil. Den andre er for at vi alle skulle være med på utviklingen. Hvert gruppemedlem følger med på utviklingen for å kunne gi tilbakemelding før koden ble flettet inn til hovedgrenen. Siste er å teste koden når ny funksjonalitet ble innført. Dette innebærer også å sjekke at gammel kode fungerte med ny kode. God kodekvalitet var viktig for gruppen fra starten av. Derfor ble disse kravene satt opp tidlig i utviklingen. De tre hovedgruppene av kvalitettesting blir forklart nærmere i følgende seksjoner.

En annen viktig grunn for regelmessig testing var fordi løsningen ble utviklet som fire separate moduler. Hvert av disse skulle snakke sammen til slutt. Ved å innføre diverse verktøy kunne en forsikre seg på best mulig måte at det ikke ble kræsj mellom modulene.

10.1 Kodekvalitet

Kvalitet på kode er viktig for å holde en ryddig kodebase. Om hver utvikler har sin egen programmeringsstil, eller rekkefølge på importeringer, blir koden fort rotete. Denne seksjonen vil beskrive de diverse verktøyene som ble brukt for å sørge for at all kode holdt samme stil.

10.1.1 Linting/typesjekker

Et verktøy som kan brukes for å ta syntaksfeil tidlig er bruk av teksteditorer som støtter linting, eller statisk kodeanalyse (Static program analyser). Dette betyr at koden blir sjekket for eventuelle programmatiske feil eller stilbrudd. Fordelen her er at man ikke trenger å kjøre koden eller skrive tester som sjekker den. Verktøyet sjekker koden opp mot en rekke regler. Figur 10.1 viser varsel av importeringer som ikke blir brukt. Det er sånne feil som ofte kan bli oversett ved at man ikke har noe visuelt som påminner. Selvfølgelig er det oss som utviklere som skal passe

Figur 10.1: Linter feil gitt fra Pyright.

på dette, men et verktøy som sjekker dette frigjør tid som utvikler kan bruke på andre ting.

Verktøyet som ble brukt av gruppen heter Pyright[1]. Det finnes andre som for eksempel flake8[2], mypy[3], pylint[4], o.l. Gruppen mente Pyright var det rette verktøyet da den inneholdt de funksjonene som gruppen var ute etter. De funksjonene som ble brukt var statisk kodesjekk, typesjekk og se etter overflødig kode, Python har i utgangspunktet et dynamisk type system, der det ikke trengs å gi variabler en type før den brukes. Fra Python versjon 3.5 tilbys det offisiell støtte for *type hinting* [5], som kan brukes av eksterne verktøy som Pyright for å validere typene til variabler. Dette er ikke brukt under kjøring av programmet, kun kjørt under utviklingen.

En annen grunn er at Pyright også kan fungere som en språktjener[6]. Det vil si at Pyright kjører i bakgrunnen og for hver endring som gjøres i koden, blir endringene sjekket opp mot Pyright. Dette foregår i sanntid og medfører at feil kan tas tidlig. Dette systemet ble konfigurert hos alle utviklere.

10.1.2 Formatering/stil

Det som ofte kan være problem i et team av utviklere er at alle kan ha hver sin måte/stil å skrive kode på. For eksempel kan dette være mellomrom mellom parentes og krøllparentes, blanke linjer mellom deler av koden, og lignende. Når koden skal flettes inn til en felles gren i kodebrønnen, er det til fordel at stilen er lik. Hvis ikke kan det risikeres at mer enn ønsket kode blir endret på grunn av en spesifikk tekstbehandlers autoformateringsverktøy. Dette kan gjøre endringslogger uklare, ved at mer enn ønsket er endret. Det vil også kunne bidra til flettekonflikt på uventede plasser i koden som ikke har en enkel og naturlig løsning. Så lenge koden inneholder lik stil, skaper det minst mulig ulemper for gruppen. Derfor vil koden til ny funksjonalitet ha et større fokus under en kodeanmeldelse. For å ivareta kodestilen mellom gruppemedlemmene ble black[7], isort[8], prettier[9] og pydocstyle[10].

[1] `https://github.com/microsoft/pyright/`, besøkt 07.05.2021
[2] `https://flake8.pycqa.org/en/latest/`, besøkt 12.05.2021
[3] `http://mypy-lang.org`, besøkt 12.05.2021
[4] `http://pylint.org`, besøkt 11.05.2021
[5] `https://docs.python.org/3.5/whatsnew/3.5.html#whatsnew-pep-484`, besøkt 12.05.2021
[6] `https://microsoft.github.io/language-server-protocol/`, besøkt 11.05.2021
[7] `https://black.readthedocs.io/`, besøkt 07.05.2021
[8] `https://pycqa.github.io/isort/`, besøkt 07.05.2021
[9] `https://prettier.io`, besøkt 07.05.2021
[10] `http://www.pydocstyle.org/`, besøkt 07.05.2021

black er en generell stilformatter for Python. Denne passer på at alt er innenfor bestemt linjelengde, like mange linjeskift mellom funksjoner og fjerning av whitespace som er overflødig.

isort er et supplement til black for å holde importeringer sortert logisk; systemmoduler på topp, 3. parts biblioteker, og til slutt våre egne moduler. Dette er igjen med på å få en mer tydelig struktur på filene. Når importene er i en logisk rekkefølge er det enkelt å se hva som kommer fra hvor.

pydocstyle er for å sørge for at Python-koden er dokumentert. Det ble valgt å følge NumPy[11] sin kodestil. Denne har gruppen erfaring med fra tidligere og mente at det ville være misbruk av ressurser om gruppen skulle lære en ny stil på nytt. Samtidig er stilen ryddig og oversiktlig.

prettier har hovedsaklig blitt brukt for frontend. I motsetning til black, som er stilformatter for Python. Er prettier laget for å stilformattere HTML/JavaScript/CSS/YAML.

For å sørge for at alle utviklere bruker verktøyene, har gruppen tatt i bruk *pre-commit*[12] som et et overordnet verktøy. Det *pre-commit* gjør er å legge seg til som en hook til Git. Slik at når ny kode prøves å *commites* til Git, kjøres alle sjekkene over på de filene som er sist endret.

10.2 Samarbeid

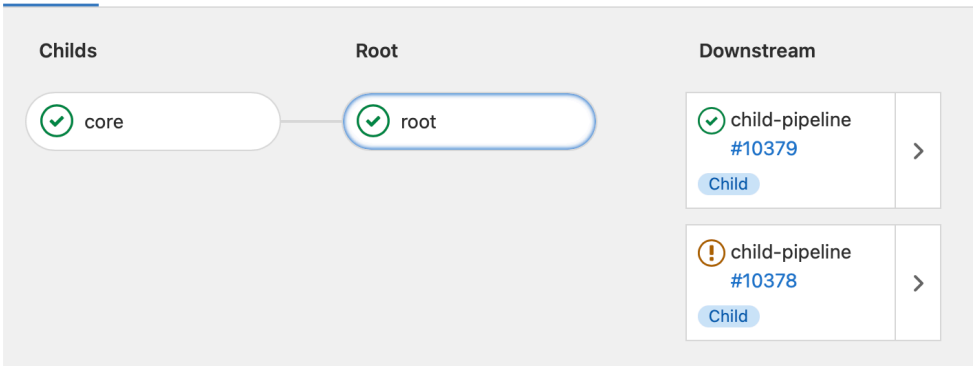
GitLab er en samling av tjenester som utgjør en komplett løsning som gjør det enklere å jobbe sammen. Under hele utviklingen jobbet gruppen opp mot institutts GitLab-instans[13]. Instansen ble brukt som kodebrønn for rapporten og koden, som Kanban-brett og kjøring av testing ved hjelp av pipelines.

Utviklingen foregikk på hver gruppemedlems egen maskin. Når ny skulle bli flettet inn mot *master* ble det benyttet en pipeline for å kjøre en rekke tester. Ved fletteanmodninger ble en mal brukt for å sørge for at utvikleren beskrev hva den nye koden skulle prøve å løse.

10.2.1 Fletteanmodninger

Arbeidsflytet for hver fletteanmodning var å lage en egen gren, dytte denne opp til kodebrønnen for så å åpne en fletteanmodning. Når en ny anmodning lages vil en bli møtt av en ferdigutfylt mal som inneholder en rekke punkter som må fylles ut. Kodeliste 10.1 viser denne malen. Denne er delt inn i flere deler som skal hjelpe i å få en rask oversikt over hva denne fletteanmodningen ønsket å legge til eller endre. Her ble det lagt opp til å legge til en beskrivende tekst, sjekkliste om nye tester eller dokumentasjon er lagt til og resultat fra kodekvalitets verktøyene.

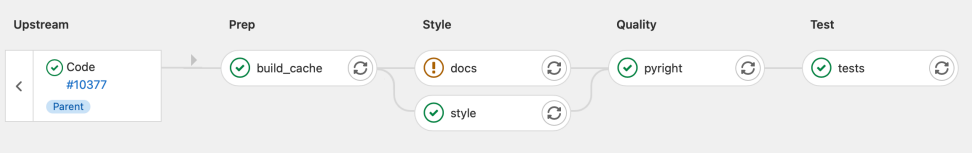
[11] `https://numpydoc.readthedocs.io/`, besøkt 07.05.2021
[12] `https://pre-commit.com`, besøkt 19.05.2021
[13] `https://git.gvk.idi.ntnu.no`, besøkt 12.05.2021



Figur 10.2: Pipelines set fra roten av prosjektet.

```
### Summary

<!-- Summarize the merge request encountered concisely. -->

### What is the current *bug* behavior?

<!-- Describe what actually happens. -->

### What is the expected *correct* behavior?

<!-- Describe what you should see instead. -->

### Checklist

<!-- Pick the applicable tasks -->

- [ ] Tests
- [ ] Docstrings

### Output of checks

<!-- black/pyright -->

#### Environment details

<!--  Input any relevant environment information if needed. -->

### Possible fixes

<!-- What is this merge request attempting as solving. -->
```

Kodeliste 10.1: Mal for fletteanmodninger



Figur 10.3: Stegene i en child-pipeline.

10.2.2 Pipelines

Pipelines var delt inn slik at den hadde en rot-pipeline og flere child-pipelines. Se figur 10.2 for rot og figur 10.3 for en child-pipeline. Som en child-job viser, er det her alle de individuelle sjekkene kjører. Ut i fra figuren ses de forskjellige stegene som titler på toppen av kolonnene. Disse kjøres i en logisk rekkefølge angitt av stegene i konfigurasjon.

Vedlegg N viser flyten i en child-pipeline. Oppsettet er delt inn i to deler. Første del handler om det generiske som omhandler hver jobb. Andre halvdel deklarerer de individuelle jobbene.

Første del starter med å sette hvilke operativsystem/containerbilde som skal benyttes. Deretter settes stegene som skal kjøres. Her er det viktig å være klar over at om et steg ikke er velykket, kan dette medføre at neste steg ikke kan kjøre. Stegene samsvarer med figur 10.3. Neste er å sette variabler. De kan enten være variabler som brukes i selve filen eller inne i jobbene. Til slutt settes det opp et system for mellomlagring. Siden de fleste jobbene deler de samme avhengighetene, er det nyttig å lagre de unna. Kalkulering av avhengigheter gjøres i første jobb.

Andre halvdel handler om de ulike jobbene som utgjør de jobbene/testene som kjøres hver gang kode til dyttet til en flette anmodning. Med at koden er hovedsaklig skrevet i Python, kan det bli en del avhengigheter, første jobb er da med på å sørge for at disse avhengigheten blir lastet ned og tatt vare på videre. Dette skjer som en del av mellomlagringen. Neste er å sjekke koden for eventuelle stilbrudd og programmatiske feil. Her kjøres noen av verktøyen som nevnt i avsnitt 10.1.2. Og til slutt kjøres alle testene på koden som fletteanmodningen berører.

10.3 Testing

Pytest, se avsnitt 4.2.3, benyttes for integrasjon- og enhetstester. Inkludert i pytest er det en *decorator*, `pytest.fixture`. Denne kan legges på en funksjon for å registrere den som en *fixture*. Dette kan så legges inn som parameter til en testfunksjon, og koden i denne *fixutur*-en blir så kjørt før en test [60]. Dette kan brukes til å initiere testdata, slik at alle disse er like for alle testene som trenger det. Kodeliste 10.2 viser en slik *fixture*, her opprettes en *sqlite*-database som da kan benyttes i testene. Det er viktig å påpeke at en *fixture* ikke blir tilbakestilt etter en test som bruker den er ferdig. Som konsekvens vil databasen ikke bli rensket mellom hver test som

benytter den.

```
@pytest.fixture
def in_memory_sqlite_db():
    """Create a sqlite database in memory for use with tests."""
    engine = create_engine("sqlite:///:memory:")
    metadata.create_all(engine)
    return engine
```

Kodeliste 10.2: Eksempel av en Pytest *fixture*

10.3.1 Enhetstesting

Python er et tolket språk og gir ikke mulighet for å validere koden før den blir kjørt. Dette kan føre til økt mengde kjøretidsfeil kontra kompilerte språk som for eksempel C++ eller Rust, hvor disse kan fanges når koden kompileres. Konsekvensen av tolkede språk kan for eksempel være at en avgrening i en `if`-setning kjøres kun i veldig spesielle tilfeller. I denne grenen kan det ligge ett funksjonskall til en funksjon som forventer en gitt datatype, for eksempel om den forventer en liste og derfor bruker metoder å operere på lister internt, men et nummer blir feilaktig sendt istedenfor. Denne feilen kan ligge uoppdaget lenge, og en sjelden gang kræsje programmet. Ved å benytte enhetstester som dekker hele kodebasen kan slike feil oppdages og mitigeres før kjøring av programmet.

En enhetstest kan lages for alt fra en funksjon til en klasse. Kodeliste 10.3 viser en enhetstest. Denne tester funksjonaliteten til et objekt som endrer dens klassifisering etter hvert som den får nye deteksjoner lagt på seg. Den benytter også en *fixture*, `make_test_obj`, som er en funksjon som returnerer en liste med ferdige objekt.

Kodeliste 10.3 sjekker først at objektet ser ut som forventet. Dette er svært viktig dersom det benyttes en *fixture*, da ting kan endres. I en test er det viktig at all data er kjent når den blir laget. Etter dette er sjekket endres objektet, og det blir sjekket igjen. Denne gang for å se at ønsket parameter ga ønsket resultat.

```python
@pytest.mark.usefixtures("make_test_obj")

def test_calc_label(make_test_obj: List[Object]):
    """Test calculating label label and probability."""
    obj = make_test_obj[0]

    assert round(obj.probability, 2) == 0.4
    assert obj.label == 1
    assert isinstance(obj.label, int)

    obj.add_detection(Detection(BBox(*[25, 35, 45, 55]), 0.5, 2, 4))
    obj.add_detection(Detection(BBox(*[25, 35, 45, 55]), 0.3, 2, 4))
    obj.add_detection(Detection(BBox(*[25, 35, 45, 55]), 0.8, 2, 4))

    assert round(obj.probability, 3) == 0.343
    assert obj.label == 2
```

Kodeliste 10.3: Test av et objekts klassifiseringsalgoritme

10.3.2 Integrasjonstesting

Dette fungerer hovedsaklig på samme måte som en enhetstest, se avsnitt 10.3.1. Forskjellen er "nivået". En integrasjonstest sjekker grensesnittet mellom funksjoner/klasser/moduler. Dette ble benyttet for å sjekke API-endepunktene, for eksempel når en jobb startes, at den også blir ferdig. Denne type testing inkluderer ofte funksjonalitet fra flere pakker i løsningen, og kan derfor ikke implementeres som en enhetstest.

10.3.3 Brukertesting

Brukertest handler om å teste hele løsningen, sjekke om de som skal benytte forstår hvordan alt henger sammen. Dette inngår også under testkategorien *akkseptanstesting*, som handler om å teste programmet fra start til slutt for å verifisere at programmet fungerer som et svar på problemstillingen. Brukertest av løsningen går ut på å sjekke hele prosessen fra å opprette et prosjekt til å eksportere resultat.

For å gjennomføre brukertest ble et spørreskjema opprettet, se vedlegg E. Her blir testpersonen møtt med en oppgave. Når oppgaven er gjennomført skal testpersonen svare på relaterte spørsmål. Eksempel på en slik instruksjon er "Lag et prosjekt med navn, prosjektnummer og beskriving.". Utover dette skal de ikke hjelpes. Dette er for at testpersonen selv skal resonere seg fram til hvordan bruke grensesnittet.

På grunn av *COVID*-restriksjonene blir det lagt opp til å gjennomføre testen over *Zoom*. Hver testperson blir tildelt eget *breakout room* slik at personen kan sitte uforstyrret. Sammen med testpersonen skal det medfølge minst en utvikler som observatør for å bistå med teknisk støtte dersom det vil være behov. Forsøket

skal etter avtale med testperson bli gjort opptak av skjerm. Dette er for at gruppen skal kunne ha opptaket som referanse dersom noe skulle være uklart etter endt test. Opptaket skal gjøres anonymt ved å ikke inkludere testpersonens navn eller ansikt. Dette vil basert på NTNU sine bestemmelser ikke være definert som personvernsopplysning [61], og ingen ytterligere tiltak må iverksettes for å beskytte personen.

Kapittel 11

Distribusjon

For at oppdragsgiver og andre skal lett kunne ta i bruk løsningen. Trengs det en måte å samle sammen de ulike modulene på en sånn måte at det ikke er utfordrene å komme i gang. Valget av arkitektur stilte ekstra krav. Dette var hvordan de ulike modulene var satt sammen og utviklet hver for seg. Med målet om at løsningen kunne kjøres distribuert gjorde at modulene ble betrakter som ulike deler. Sammenlignet om alt ble kodet i samme pakke, måtte det nå sørges for at alle ble inkludert og kan snakke sammen.

Det ble allerede fra start bestemt å bruke et pakkeverktøy som heter Poetry, se avsnitt 4.2.4, på grunn av funksjonaliteten den gir for utviklingsprosessen. Dette verktøyet skulle være med å gjøre det enklere å pakke de forskjellige modulene slik at de var enklere å distribuere. Poetry sin rolle er å se til at pakken inneholder de nødvendige definisjonene som navn, versjonsnummer, avhengigheter og annen metadata. Dette er med på gjøre det mulig å installere pakken på en annen maskin uten å hente ned koden manuelt fra kodebrønnen. Bruken i gruppens løsning var tenkt at hver modul skulle ble pakket av Poetry slik at det var lettere å installere de andre steder.

For enkel utrulling ble det vurdert en kontainer-basert løsning. med en slik løsning var det mulig å pakke alle modulene i ett bilde eller et bilde per modul. Fordelen med det siste bygger på ønsket om at løsningen skulle være skalerbart. Ved at disse er separerte kan man kjøre opp for eksempel flere kontainere av deteksjon og tracing slik at dette kunne forbedre ytelsen av løsningen. Ulempen her er at for å ta i bruk kontainere må man installere er 3.-partsverktøy. Det var nødvendigvis ikke ønskelig å kreve at oppdragsgiver skulle gjøre dette. Samtidig var det ikke sikkert hva som det var tillatt å installere på arbeidsstasjonene.

En annet metode gruppen så på var å pakke modulene sammen i et kjørbart program som oppdragsgiver kunne enkelt starte på sin maskin. Dette ville forenkle hvordan man kjørte løsningen ved at det ikke trengtes mer for å starte. Ved hjelp av en slik metode ville alt kunne være pakket sammen med de nødvendige bibliotekene som løsningen krever og dermed ta bort kravet om andre avhengigheter. Som for eksempel bruk av kontainere krever.

Det ble ikke lagt av tid til å gjennomføre dette på grunn av at utviklingen

foregikk helt mot siste slutt. Og gruppen hadde ikke satt av tid til å diskutere eller lage en rutine for hvordan løsningen skulle bli distribuert.

Kapittel 12

Diskusjon

12.1 Arbeidstid og -miljø

Gruppen følte at tiden ikke strakk til mot slutten av prosjektet. Dette kan skyldes manglende erfaring i planlegging av komplekse system. I planleggingsfasen ble det lagt opp til et relativt komplisert system, med en arkitektur gruppen ikke hadde erfaring med. En annen årsak til liten tid kan også være at gruppen mangler omtrent ett månedsverk med arbeidstid, se vedlegg H. Per gruppereglene ble det avtalt at hvert medlem skulle bruke 600 timer over 20 uker.

En annen faktor for redusert arbeidstid kan være at det aldri ble avtalt hvor spesifikt tidsbruk skulle føres. Mye av dette kan skyldes lite erfaring med konseptet. De fleste i gruppen har ført timer ut fra den saken de har jobbet på ved bruk av integrasjon mellom GitLab og tidstakningssystemet. Mens andre har ført timer manuelt. I noen tilfeller har dette medført at registreringen har blitt glemt. Mye av forskjellen her skyldes forskjellig nivå av integrasjon mot de spesifikke verkøyene hos medlemmene. På grunn av manglene her er det vanskelig å si hvor mye arbeid har blitt brukt på spesifikke deler av prosjektet.

Denne skevregistreringen kunne vært unngått dersom en hadde benyttet verktøy som kunne alarmert om timeantall ikke ble overholt. Det burde også vært laget en rutine på hvordan timer skulle føres. Et annet tiltak kunne vært å øke kjernetid fra kl. 9-12 til kl. 9-15. Dette kunne derimot overlappe med andre fag som gikk samtidig som prosjektet.

På grunn av restriksjonene rundt *COVID-19* var store deler av semesteret uten fysisk undervisning. Dette medførte at gruppen ikke hadde direkte behov for å møte opp på campus, som igjen resulterte med at gruppen sjeldent benyttet de fysiske lokalene som var periodevis tilgjengelig på Campus. Enkelte av gruppemedlemmene har nevnt at dette har bidratt til mindre arbeidsmotivasjon, som har vært et grunnlag for redusert arbeidstid. For andre medlemmer har det fungert best med hjemmekontor. Dette på grunnlag av bedre arbeidsmiljø med stasjonær datamaskin, ekstra skjermer og ergonomisk utstyr. Gruppen skulle hatt en bedre balanse. Det burde ha vært lagt opp til mer fysisk oppmøte.

12.2 Møter

Møte med veileder ble godt utnyttet gjennom hele prosjektet. Marius har tidligere veiledet andre prosjekt knyttet mot oppdragsgiver og bakgrunn innen datasyn og maskinlæring. På disse møtene fikk gruppen gode svar på spørsmål rundt maskinlæringsmetoder og -teknologier. Senere ut i prosjektet ble det mindre fokus rundt maskinlæring, og mer om generell utvikling. Her føltes det ikke at veileder kunne bidra like mye. Det var derimot en god plass å dele andre tanker og vise fram progresjonen og diskutere problemstillinger som måtte dukke opp. Under rapportskrivingen ble veileder aktivt benyttet for å forbedre struktur og språk.

Møter med oppdragsgiver fungerte som milepæler. Disse ble gjort ved slutten av en iterasjon og tvang gruppen til å ferdigstille funksjonalitet slik at dette kunne presenteres. Her fikk gruppen tilbakemeldinger på arbeid slik at det var mulig å gjøre endringer og sørge for at løsningen var i henhold til oppdragsgivers behov. Disse møtene bidro også med uvurderlig erfaring om kommunikasjon med kunder med mindre teknisk kompetanse. Metoder og funksjonalitet måtte forklares ekstra og progresjon måtte presenteres på en tilpasset måte.

Det ble sjeldent avtalt på forhånd hvile ny funksjonalitet som skulle vises, og heller ikke hvordan. Dette bidro til relativt uformelle møter, noe som ga en plattform for diskusjon og synsing mellom gruppen og oppdragsgiver. Det ble ikke følt at møtene var mindre konstruktive på grunnlag av formaliteten. Det var heller ikke møter der ny funksjonalitet ikke ble fremvist.

12.3 Versjonskontroll og Kvalitetssikring

Bruken av GitLab for kodebrønn har fungert godt. Det ble derimot ofte situasjoner der fletteanmodninger ble for store. Til tider var det vanskelig å gi omtale på arbeid andre hadde gjort basert på størrelse og endringsomfang. Omtaleprosessen med to godkjenninger og omfattende kvalitetskontroll gjennom GitLab sine systemer fungerte bra. Medlemmene fikk dele sine meninger på løsningene og det var enkelt å gi endringsforslag. GitLab gjorde det enkelt å referere til andre endringslogger og Kanban-kort. Referering til kortene ga automatikk med å lukke dem, og progresjon kunne enklere følges. Bruken av KanBan-brett i GitLab var også veldig fordelaktig.

Kvalitetskontrollen benyttet under utvikling var bestemt fra starten av å være restriktiv. Dette fungerte bra og gruppen erfarte aldri problemer med fletting. Dersom fletting inneholdt manglende formatering eller programfeil, vil systemet avvise endringene. Kvalitetskontrollen kunne i verste fall ta 30 minutter. Dette var ofte grunnlag for frustrasjon ved små endringer før en kunne flette. Mye tid ble også brukt på å sette opp og feilsøke systemene rundt dette. Restriksjonene og omfanget av kvalitetskontrollen ble ikke endret. Det fungerte bra jevnt over, og sørget for at koden som ble flettet ville fungere. Det vil også bidra for videreutvikling, siden nye utviklere har en kvalitetstandard de må følge. Siden kodebasen blir åpen kildekode, er det spesielt viktig med automatiske kvalitetsjekker.

Det var en hendelse der en av gruppens medlem forsøkte å rydde opp i grenene lokalt på deres maskin og med uhell slettet alle grenene på GitLab. Ingen arbeid ble tapt grunnet dette, da Git fungerer distribuert. Andre medlem hadde sine lokale kopier de påvirkede grenene. Et annet tilfelle hendte under rapportskrivingen når et annet medlem slettet mappen som inneholdt rapporten. Igjen var Git til redningen ved at var mulig å hente tilbake rapporten. Samt de siste endringene var allerede dyttet opp til kodebrønnen. Git som versjonskontrollsystem fungerte svært bra for gruppen ellers.

12.4 Rapport

Gruppen var for dårlig til å skrive rapport underveis. Dette resulterte i ekstra arbeid mot slutten av prosjektet for å få alt ferdigstill. Førebyggende tiltak mot dette skulle vært å avslutte koding tidligere. Om koding ble avsluttet tidligere ville ikke løsningen være funksjonelt komplett, som ville resultert i et dårlig sluttprodukt for oppdragsgiver.

12.5 Løsning

Oppdragsgiver ønsket et enkelt program som kunne kjøres på en enkel arbeidsstasjon. Basert på maskinene de ønsket å bruke og mengden data måtte det legges fokus i ytelse. Løsningen som er utviklet av dette prosjektet har tatt i bruk konsepter som motvirker disse ønskene.

Mikroservice-arkitekturen gjør at data i minne må konverteres for å sendes over HTTP. Dette er mindre effektivt enn å bare benytte data fra minnet direkte. Arkitekturen kompliserer også utrullingen. Fire separate program må startes, og kommunikasjon mellom enkelte må eksplisitt tillates. Et skript samler alle disse programmene som ett, som gjør oppstart enklere.

Arkitekturen tilbyr muligheten for å kjøre distribuert. De forskjellige pakkene har behov for forskjellige typer ressurser. Kjernen og sporing er avhengig av CPU, objektdeteksjon er avhengig av grafikkytelse. En GPU-optimalisert server hos Linode koster fra $1000[1] i måneden, sammenlignet med en CPU-optimalisert server til $120[2] dollar [62]. En kunne eksempelvis kjørt en CPU-optimalisert server som tjener grensesnitt og kjernen, og ved behov kjøre opp en server med GPU for objektdeteksjon og sporing. En ekstra utviding på dette kunne vært at grensesnittet kjører på en billigere server, $5 månedlig med ekstra logikk for å spinne opp serverer for de andre pakkene ved behov. En omstrukturering for å minske mengden kompleksitet kunne vært at sporing var i same pakke som deteksjon. De fleste GPU-optimaliserte serverer har også mye ram og CPU, så å kjøre sporing burde ikke være et problem.

[1] 8 CPU, 32GB RAM, 640GB SSD
[2] 8 CPU, 16GB RAM, 320GB SSD

En mer monolittisk arkitektur kunne blitt benyttet, og kanskje resultert i et mer ferdig produkt tidligere. Her ville alle pakkene vært samlet på et sted og det ville ikke vært behov for API over HTTP. Ved å samle alt ville utrulling vært enklere, da løsningen ville fungert som et enkelt program med ett inngangspunkt. Uten bruk av et HTTP-API ville det vært mindre av ytelsesproblematikken med å sende data mellom pakkene. Feilhandtering ville også vært enklere å handtere, dersom noe kræsjet ville ikke alt slutte samtidig. Dette sammen med ett inngangpunkt ville gjort det enkelt å restarte løsningen dersom feil skulle oppstå. Denne arkitekturen vil ikke nødvendigvis fungere så bra i skyen. En enkel server blir benyttet, og denne trenger grafikkytelse.

Om løsningen brukes på lokal maskinvare vil ikke mikroservice argumentasjonene være like sterke som monolittisk. En initiell investering på en krafig datamaskin til 20-30 tusen vil ikke nødvendigvis være like rask som en i skyen, men til prisen av 3 måneder for en GPU-optimalisert server så har en mulighet for å analysere i tre til fem år[3].

Det ble ikke prioritert å implementere konfigurasjonsfiler. Det er ikke mulig å spesifisere hvilken port en spesifikk pakke skal kjøres på, sette størrelsen på arbeidsmengde, *batch size*, eller bestemme hvilke mappe programmet skal hente videofiler fra uten at det endres i koden. Arbeidsmengden vil justere seg selv slik at der er plass i videominne, og rapportere feilmeldinger, men disse har liten hensikt uten at brukeren enkelt kan endre dette. Med konfigurasjonfiler kunne det også enkelt legges til rette for å kunne endre elementer i brukergrensesnitt som eksempelvis logo.

For å sørge for høy ytelse ble det gjennomført tiltak beskrevet i avsnitt 8.7. På en kraftig nok maskin er flaskehalsen stort sett deteksjonen. Dette er tid som kunne benyttes til å klargjøre neste *batch*.

SORT og YOLO var gode valg for å ivareta ytelsen. En del justeringer trengs enda her for å sikre best mulig resultat. Isolert vises de å være gode nok, men kombinert vises det at SORT ikke helt klarer å holde sporingen lenge nok.

En funksjonalitet som ble luftet for oppdragsgiver var en mulighet for dem til å kunne korrigere en deteksjon sin klasse i brukegrensesnittet. Dette ble det ikke tid til, men sees på som en funksjonalitet som ville vært nyttig. Da ville forskere hos oppdragsgiver kunne gi ekspertkunnskap ved korrigering, som kunne vært brukt til å generere nye datasett for forbedring av deteksjonsmodellen. Løsningen vil også kunne hjelpe med dette i dag, men som en mer manuell prosess.

Videre trengs det implementering av konfigurasjonfiler, justering av sporings- og maskinlæringsalgoritme og opprydding i kodebasen. Mer funksjonalitet for å utnytte den distribuerte arkitekturen ønskes også.

[3]Basert på ordinær garanti- eller serviceavtaler

Kapittel 13

Konklusjon

Oppgaven gikk ut på å lage en løsning for deteksjon av arter i undervannsforhold for Norsk institutt for naturforskning (NINA). Løsningen skulle jobbe på videomateriell samlet fra kamera de har stående ute på forskjellige lokasjoner. Oppgaven hadde som mål å automatisere denne deteksjonen og redusere tiden det tar å gjøres manuelt. For å enkelt bruke løsningen ønsket de i tillegg et brukergrensesnitt som var enkelt å bruke. I dette kapittelet går gruppen gjennom målene til oppgaven, og videre arbeid.

13.1 Måloppnåelse

Løsningen gjør det mulig for oppdragsgiver å oppdage, artsbestemme og spore individuelle undervannsarter over flere videoer. Dette oppfyller resultatmålet over hovedfunksjonaliteten til programmet beskrevet i avsnitt 1.3.2. Løsningen vil automatisk prosessere videoer i jobber ut fra brukers ønsker. Etter prosesseringen er fullført kan brukeren hente ut oppdaget objekter fra brukergrensesnittet.

Artsbestemming med ferdig opptrent modell havnet på en $mAP = 0.95$ som vist i avsnitt 6.4.2. Dette ligger godt innenfor målet på 0.6 bestemt i avsnitt 1.3.2.

Objektsporing er innenfor resultatmålene med bruk av annotert data fra treningssett. Det antas at resultatet for sporing ikke endrer seg mye med reelle deteksjoner. Resultatene fra avsnitt 7.2 viser $MOTP = 0.94$ og $MOTA = 0.98$.

Menneskelig tid er redusert betydelig, og er innenfor resultatmålene i avsnitt 1.3.2. For å starte prosesseringen brukes det noen få minutter til å definere en jobb og velge videoer. Når jobben kjøres vil objektdeteksjon utføres automatisk. Menneskelig tid har blitt erstattet med maskintid. Prosessering av en video på 30 minutter tar omtrent 40 minutter med maskinen beskrevet i tabell 6.2. Gruppen hadde et uformelt mål om å nå sanntids prosessering av videoer. Noe som var nesten oppnådd, da prosesseringen bruker omtrent 33 % lengre tid.

Et annet resultatmål var at brukergrensesnittet skal være intuitivt og enkelt å bruke. Dette har løsningen oppnådd basert på resultatene av brukertest, se vedlegg E. Oppdragsgiver var godt fornøyd med hvordan brukergrensesnittet endte

opp. Det er også lagt opp for oppdragsgiver å manuelt verifisere resultatene til en jobb. Jobben forteller blant annet når en fisk var oppdaget, art og hvilken videofil. Med denne informasjonen kan NINA velge å bruke litt mer menneskelig tid for å verifisere resultatene. For å gjøre den manuelle verifiseringen lettere, ble det implementert funksjonalitet for å se en kort videosnutt av oppdaget objekt.

Resultatet av en ferdig prosessert jobb kan eksporteres som en CSV-fil. Dette oppnår resultatmål om at løsning skal produsere en rapport. Filen inneholder all informasjon tilknyttet jobben. Eksporteringen av data er et viktig verktøy for NINA til å kunne bearbeide forskningsdata. På den måten er ikke de begrenset til visningsmetoder gruppen implementerer i brukergrensesnittet. CSV er også et filformat som er godt støttet av forskjellige program. Eksempelvis Microsoft Excel, eller programmeringspråket R.

13.2 Videre arbeid

Det som gruppen ønsker mest å jobbe videre med er å strukturere kodebasen bedre. Slik som kodebasen er ved oppgaveslutt, er det mye kode som kan omskrives for å bedre logikk og ryddes opp i. Dette gjør det lettere for nye utviklere å sette seg inn i hvordan løsningen fungerer. Siden løsningen skal publiseres som åpen kildekode må det legges til rette for at andre ønsker å fortsette arbeidet. Utvikling av ny funksjonalitet er mer utfordrende med en rotete kodebase. Derfor er dette gruppens viktigste punkt, siden det påvirker alt videre arbeid.

Løsningen var implementert fleksibel nok til å gi mulighet å utvide med deteksjon av andre arter. Om en endrer underliggende modell vil den kunne detektere andre typer objekter. For eksempel har NINA nevnt mulighet med å detektere skogsdyr fanget opp på viltkamera. Løsningen er nødvendig vis ikke begrenset til å detektere undervannsarter, men et nytt datasett må lages. Som et resultat ser gruppen mange muligheter for å utvide løsningen gjennom senere bacheloroppgaver. I tillegg er koden utgitt som åpen kildekode hvor alle som ønsker kan bidra. Publisering av løsningen kan bety at forskere verden over vil lettere kunne jobbe med forskning på objekter i video.

Konfigurasjonsfiler er noe gruppen ønsket å ta seg tid til. Filene kan brukes til å kjøre løsningen distribuert over flere maskiner. Dette er fordi API-kommunikasjon mellom core, detection og `tracing` bruker HTTP-protokollen. For eksempel kan en kjøre hele løsningen på en server. Dermed kan en bruke applikasjonen over et helt nettverk. Til dette bruksområdet burde sikkerheten forbedres. Konfigurasjonsfiler vil også gjøre det lettere for brukere å tilpasse løsningen til slik de ønsker. Et eksempel på dette kan være å endre til sin egen logo, i stedet for NINA sin. Andre bruksområder kan være å finjustere parametre i systemet, eksempelvis batchstørrelse.

Forbedret ytelse er noe som har blitt vurdert å gjøre noe med. Her er det mange muligheter. Den enkleste vil være å bruke de kraftigeste maskinspesifikasjonene, men dette lar seg ikke skalere. Derfor kan en mulig utvidelse være å kunne kjøre flere objekt detektorer på flere maskiner. Gjennom en IaC “Infrastructure as Code”

plattform kan en ha mange maskiner tilgjengelig. Om core modulen kunne distribuere arbeid mellom arbeidsmaskiner, vil deteksjons ytelsen kunne skalere basert på arbeidsmengden.

Feilhåndtering burde forbedres gjennom hele løsningen. Selv med tester av funksjonalitet er det tilfeller hvor uventet kan skje. For eksempel om et API ikke kan kontaktes, eller prosesseringen kræsjer. Bruker får ikke varsel om at dette gjennom brukergrensesnittet. En er avhengig av løsningens meldinger til kommandolinjen for å oppdage feil. Fra starten burde det ha vært planlagt et system for å varsle bruker om noe var galt, eller i det minste håndtere feilene automatisk.

Administrasjonsside for løsningen var ønskelig å implementere. Grunnet for lite tid ble dette nedprioritert. Som del av videre arbeid kan denne funksjonaliteten implementeres. På siden kan brukeren eksempelvis administrere systemparametere til løsningen. En kan også bruke siden for senere å implementere bruker autentisering og administrasjon.

Modell fra maskinlæring er også noe som kan utvides og forbedres. Datasettet gruppen hadde tilgjengelig fra NINA var fra en lokasjon. Modellen gruppen produserte vil nok ikke klare å gjenkjenne arter fra ulike miljøer like godt. For eksempel fra en annen kameravinkel, eller annen bakgrunn. Det er også usikkerhet hva som skjer om en ukjent art blir oppdaget, noe som kan resultere i feil. Derfor kan det være gunstig å utvide datasettet med flere arter, fra flere lokasjoner.

Bibliografi

[1] I. Goodfellow, Y. Bengio og A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[2] T. H. Holter, K. M. Myrvold, U. Pulg og J. Museth, «Evaluating a fishway reconstruction amidst fluctuating abundances,» *River Research and Applications*, årg. 36, nr. 8, s. 1748–1753, 2020. DOI: https://doi.org/10.1002/rra.3688. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/rra.3688. adresse: https://onlinelibrary.wiley.com/doi/abs/10.1002/rra.3688.

[3] S. A. Nørstebø og E. Myrum, *Automatisk gjenkjenning av settefisk*, nor, 2019. adresse: http://hdl.handle.net/11250/2617894.

[4] H. A. W. Haugen, *Bruk av maskinsyn for automatisk telling og artsbestemmelse av villfisk som beiter under oppdrettsmerder*, 2020. adresse: https://hdl.handle.net/11250/2664006.

[5] T. E. Eide, «Study of the Release Process of Open Source Software: Case Study,» 2007. adresse: http://hdl.handle.net/11250/251217.

[6] N. Dalal og B. Triggs, «Histograms of oriented gradients for human detection,» i *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*, Ieee, bd. 1, 2005, s. 886–893.

[7] C. Cortes og V. Vapnik, «Support-vector networks,» *Machine learning*, årg. 20, nr. 3, s. 273–297, 1995.

[8] R. Szeliski, *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010.

[9] S. Russell og P. Norvig, «Artificial intelligence: a modern approach,» 2002.

[10] T. Mitchell, *Machine Learning*, ser. McGraw-Hill International Editions. McGraw-Hill, 1997, ISBN: 9780071154673. adresse: https://books.google.no/books?id=EoYBngEACAAJ.

[11] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan og D. Hassabis, «A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play,» *Science*, årg. 362, nr. 6419, s. 1140–1144, 2018, ISSN: 0036-8075. DOI: 10.1126/science.aar6404. eprint: https://

science.sciencemag.org/content/362/6419/1140.full.pdf. adresse: https://science.sciencemag.org/content/362/6419/1140.

[12] W. Commons, *File:Colored neural network.svg — Wikimedia Commons, the free media repository*, [Online; accessed 27-April-2021], 2021. adresse: https://commons.wikimedia.org/w/index.php?title=File:Colored_neural_network.sv%20g&oldid=534702454.

[13] J. Han og C. Moraga, «The influence of the sigmoid function parameters on the speed of backpropagation learning,» i *From Natural to Artificial Neural Computation*, J. Mira og F. Sandoval, red., Berlin, Heidelberg: Springer Berlin Heidelberg, 1995, s. 195–201, ISBN: 978-3-540-49288-7.

[14] C. M. Bishop, *Pattern recognition and machine learning*. springer, 2006.

[15] W. Commons, *File:Typical cnn.png — Wikimedia Commons, the free media repository*, [Online; accessed 30-April-2021], 2021. adresse: %5Curl%7Bhttps://commons.wikimedia.org/w/index.php?title=File:Typical_cnn.png&oldid=5351%2087198%7D.

[16] J. Redmon, S. Divvala, R. Girshick og A. Farhadi, *You Only Look Once: Unified, Real-Time Object Detection*, 2016. arXiv: 1506.02640 [cs.CV].

[17] A. Bochkovskiy, C.-Y. Wang og H.-Y. M. Liao, *YOLOv4: Optimal Speed and Accuracy of Object Detection*, 2020. arXiv: 2004.10934 [cs.CV].

[18] J. Redmon og A. Farhadi, *YOLOv3: An Incremental Improvement*, 2018. arXiv: 1804.02767 [cs.CV].

[19] D. Mladeni, J. Brank og M. Grobelnik, «Document Classification,» i *Encyclopedia of Machine Learning*, C. Sammut og G. I. Webb, red. Boston, MA: Springer US, 2010, s. 289–293, ISBN: 978-0-387-30164-8. DOI: 10.1007/978-0-387-30164-8_230. adresse: https://doi.org/10.1007/978-0-387-30164-8_230.

[20] M. Hossin og M. Sulaiman, «A review on evaluation metrics for data classification evaluations,» *International Journal of Data Mining & Knowledge Management Process*, årg. 5, nr. 2, s. 1, 2015.

[21] V. S. Subramanyam, *IOU (Intersection over Union)*, Online; https://medium.com/analytics-vidhya/iou-intersection-over-union-705a39e7acef, hentet 05. Mai 2021, jan. 2021.

[22] S. Challa, M. R. Morelande, D. Muicki og R. J. Evans, *Fundamentals of Object Tracking*, eng. Cambridge: Cambridge University Press, 2011, ISBN: 0521876281.

[23] A. Bewley, Z. Ge, L. Ott, F. Ramos og B. Upcroft, «Simple Online and Realtime Tracking,» *CoRR*, årg. abs/1602.00763, 2016. arXiv: 1602.00763. adresse: http://arxiv.org/abs/1602.00763.

[24] N. Wojke, A. Bewley og D. Paulus, «Simple Online and Realtime Tracking with a Deep Association Metric,» i *2017 IEEE International Conference on Image Processing (ICIP)*, IEEE, 2017, s. 3645–3649. DOI: 10.1109/ICIP.2017.8296962.

[25] N. Wojke og A. Bewley, «Deep Cosine Metric Learning for Person Re-identification,» i *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*, IEEE, 2018, s. 748–756. DOI: 10.1109/WACV.2018.00087.

[26] K. Ulicna, G. Vallardi, G. Charras og A. R. Lowe, «Automated deep lineage tree analysis using a Bayesian single cell tracking approach,» *bioRxiv*, 2020. DOI: 10.1101/2020.09.10.276980. eprint: https://www.biorxiv.org/content/early/2020/09/10/2020.09.10.276980.full.pdf. adresse: https://www.biorxiv.org/content/early/2020/09/10/2020.09.10.276980.

[27] A. Bove, D. Gradeci, Y. Fujita, S. Banerjee, G. Charras og A. R. Lowe, «Local cellular neighborhood controls proliferation in cell competition,» *Molecular Biology of the Cell*, årg. 28, nr. 23, s. 3215–3228, 2017. DOI: 10.1091/mbc.E17-06-0368. eprint: http://www.molbiolcell.org/content/28/23/3215.full.pdf+html. adresse: http://www.molbiolcell.org/content/28/23/3215.abstract.

[28] K. Bernardin og R. Stiefelhagen, «Evaluating Multiple Object Tracking Performance: The CLEAR MOT Metrics,» *J. Image Video Process.*, årg. 2008, jan. 2008, ISSN: 1687-5176. DOI: 10.1155/2008/246309. adresse: https://doi.org/10.1155/2008/246309.

[29] P. S. Foundation, *FAQ: What is Python?* Online; accessed 01-May-2021, 2021. adresse: https://docs.python.org/3/faq/general.html%5C#what-is-python.

[30] *PostgreSQL*, Online; https://docs.sqlalchemy.org/en/14/dialects/postgresql.html, hentet 17. mai 2021, 2021.

[31] *SQLite*, Online; https://docs.sqlalchemy.org/en/14/dialects/sqlite.html, hentet 17. mai 2021, 2021.

[32] Wikipedia contributors, *Objectrelational mapping — Wikipedia, The Free Encyclopedia*, https://en.wikipedia.org/w/index.php?title=Object%E2%80%93relational_mapping&oldid=1021719270, [Online; accessed 17-May-2021], 2021.

[33] Wikipedia contributors, *Enhanced entityrelationship model — Wikipedia, The Free Encyclopedia*, https://en.wikipedia.org/w/index.php?title=Enhanced_entity%E2%80%93relationship_model&oldid=1013955453, [Online; accessed 17-May-2021], 2021.

[34] Wikipedia contributors, *Pipeline (software) — Wikipedia, The Free Encyclopedia*, https://en.wikipedia.org/w/index.php?title=Pipeline_(software)&oldid=999123151, [Online; accessed 14-May-2021], 2021.

[35] Wikipedia contributors, *Multitier architecture — Wikipedia, The Free Encyclopedia*, https://en.wikipedia.org/w/index.php?title=Multitier_architecture&oldid=1020301475, [Online; accessed 14-May-2021], 2021.

[36] Wikipedia contributors, *Service-oriented architecture — Wikipedia, The Free Encyclopedia*, https://en.wikipedia.org/w/index.php?title=Service-oriented_architecture&oldid=1021842813, [Online; accessed 14-May-2021], 2021.

[37] Wikipedia contributors, *Microservices — Wikipedia, The Free Encyclopedia*, https://en.wikipedia.org/w/index.php?title=Microservices&oldid=1021710703, [Online; accessed 14-May-2021], 2021.

[38] S. H. Shaikh, K. Saeed og N. Chaki, «Moving Object Detection Using Background Subtraction,» i *Moving Object Detection Using Background Subtraction*. Cham: Springer International Publishing, 2014, s. 15–23, ISBN: 978-3-319-07386-6. DOI: 10.1007/978-3-319-07386-6_3. adresse: https://doi.org/10.1007/978-3-319-07386-6_3.

[39] D. G. Lowe, «Object recognition from local scale-invariant features,» i *Proceedings of the seventh IEEE international conference on computer vision*, Ieee, bd. 2, 1999, s. 1150–1157.

[40] H. Bay, T. Tuytelaars og L. Van Gool, «Surf: Speeded up robust features,» i *European conference on computer vision*, Springer, 2006, s. 404–417.

[41] R. Girshick, J. Donahue, T. Darrell og J. Malik, *Rich feature hierarchies for accurate object detection and semantic segmentation*, 2014. arXiv: 1311.2524 [cs.CV].

[42] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu og A. C. Berg, «SSD: Single Shot MultiBox Detector,» *Lecture Notes in Computer Science*, s. 21–37, 2016, ISSN: 1611-3349. DOI: 10.1007/978-3-319-46448-0_2. adresse: http://dx.doi.org/10.1007/978-3-319-46448-0_2.

[43] N. OMahony, S. Campbell, A. Carvalho, S. Harapanahalli, G. V. Hernandez, L. Krpalkova, D. Riordan og J. Walsh, «Deep learning vs. traditional computer vision,» i *Science and Information Conference*, Springer, 2019, s. 128–144.

[44] *Evaluering / referanseindex for YOLOv5, modeller trent på COCO*, Online; besøkt 27.04.2021, 2021. adresse: https://github.com/ultralytics/yolov5/blob/4890499344e21950d985e1a77e84a0a4161d%201db0/README.md#pretrained-checkpoints.

[45] M. Tan, R. Pang og Q. V. Le, *EfficientDet: Scalable and Efficient Object Detection*, 2020. arXiv: 1911.09070 [cs.CV].

[46] G. Jocher, A. Stoken, J. Borovec, NanoCode012, A. Chaurasia, TaoXie, L. Changyu, A. V, Laughing, tkianai, yxNONG, A. Hogan, lorenzomammana, AlexWang1900, J. Hajek, L. Diaconu, Marc, Y. Kwon, oleg, wanghaoyang0106, Y. Defretin, A. Lohia, ml5ah, B. Milanko, B. Fineran, D. Khromov, D. Yiwei, Doug, Durgesh og F. Ingham, *ultralytics/yolov5: v5.0 - YOLOv5-P6 1280 models, AWS, Supervise.ly and YouTube integrations*, versjon v5.0, apr. 2021. DOI: 10.5281/zenodo.4679653. adresse: https://doi.org/10.5281/zenodo.4679653.

[47] B. Sekachev, N. Manovich, M. Zhiltsov, A. Zhavoronkov, D. Kalinin, B. Hoff, TOsmanov, D. Kruchinin, A. Zankevich, DmitriySidnev, M. Markelov, Johannes222, M. Chenuet, a-andre, telenachos, A. Melnikov, J. Kim, L. Ilouz, N. Glazov, Priya4607, R. Tehrani, S. Jeong, V. Skubriev, S. Yonekura, vugia truong, zliang7, lizhming og T. Truong, *openvinotoolkit/cvat: v1.2.0*, versjon v1.2.0, jan. 2021. DOI: https://doi.org/10.5281/zenodo.3497105. adresse: https://github.com/openvinotoolkit/cvat.

[48] *openvinotoolkit/datumaro: v0.1.7*, versjon v0.1.7, mar. 2021. adresse: https://github.com/openvinotoolkit/datumaro.

[49] L. Perez og J. Wang, *The Effectiveness of Data Augmentation in Image Classification using Deep Learning*, 2017. arXiv: 1712.04621 [cs.CV].

[50] M. Buda, A. Maki og M. A. Mazurowski, «A systematic study of the class imbalance problem in convolutional neural networks,» *Neural Networks*, årg. 106, s. 249–259, 2018, ISSN: 0893-6080. DOI: https://doi.org/10.1016/j.neunet.2018.07.011. adresse: https://www.sciencedirect.com/science/article/pii/S0893608018302107.

[51] *MOT20 Results*, Online; https://motchallenge.net/results/MOT20/, hentet 30. April 2021, 2020.

[52] *MOT17 Results*, Online; https://motchallenge.net/results/MOT17/, hentet 30. April 2021, 2017.

[53] C. Wootton, *Patterns of Enterprice Application Architecture, From Sprockets and Rasters to Macro Blocks*. Focal Press, 2005, ISBN: 0-240-08630-1.

[54] I. Turner-Trauring, *Reducing NumPy memory usage with lossless compression*, PythonSpeed.com, red., 2019. adresse: https://pythonspeed.com/articles/numpy-memory-footprint/.

[55] M. Fowler, *Patterns of Enterprice Application Architecture, Pattern Enterpr Applica Arch*. Addiston-Wesley, 2012, ISBN: 9788251924467.

[56] M. Sanders, *The Most Popular Python Web Frameworks in 2021*, 2020. adresse: https://scoutapm.com/blog/the-most-popular-python-web-frameworks-in-2020.

[57] D. Contributurs, *Writing your first Django app, part 1 | Django documentation | Django*, 2021. adresse: https://docs.djangoproject.com/en/3.2/intro/tutorial01//#creating-a-project.

[58] F. Contributurs, *Quickstart - Flask Documentation (2.0.x)*, 2021. adresse: https://flask.palletsprojects.com/en/2.0.x/quickstart/.

[59] D. Ghimire, «Comparative study on Python web frameworks: Flask and Django,» Metropolia University of Applied Sciences, thesis, 2020. adresse: http://urn.fi/URN:NBN:fi:amk-2020052513398.

[60] *pytest fixtures: explicit, modular, scalable*, Online; https://docs.pytest.org/en/6.2.x/fixture.html, Hentet 14. Mai 2021.

[61] *Behandle personopplysninger i student- og forskningsprosjekt*, Online; https://innsida.ntnu.no/wiki/-/wiki/Norsk/Behandle+personopplysninger+i+student-+og+forskningsprosjekt?_36_redirect=https%3A%2F%2Finnsida.ntnu.no%2Fwiki%3Fp_p_id%3D36%26p_p_lifecycle%3D0%26p_p_state%3Dnormal%26p_p_mode%3Dview%26p_p_col_id%3Dcolumn-1%26p_p_col_count%3D1%26p_r_p_185834411_title%3DBehandle%2Bpersonopplysninger%2Bi%2Bstudent-%2Bog%2Bforskningsprosjekt%26p_r_p_185834411_nodeId%3D24647%26p_r_p_185834411_nodeName%3DNorsk%26_36_redirect%3Dhttps%253A%252F%252Finnsida.ntnu.no%252Fwiki%252F-%252Fwiki%252FNorsk%252FBehandle%252Bpersonopplysninger%252Bi%252Bstudent-%252Bog%252Bforskningsprosjekt%26_36_struts_action%3D%252Fwiki%252Fview_page_history&_36_version=7.7.

[62] *Linode Pricing List*, Online; https://www.linode.com/pricing/, Hentet 18. mai 2021.

Vedlegg A

Prosjekt Avtale

NTNU

Vår dato Vår referanse

Norges teknisk-naturvitenskapelige universitet

Prosjektavtale

mellom NTNU Fakultet for informasjonsteknologi og elektroteknikk (IE) på Gjøvik (utdanningsinstitusjon), og

Norsk Institutt for Naturforskning (NINA)

(oppdragsgiver), og

Birger Johan Nordølum, Eirik Lavik, Kristian Haugen, Tom-Ruben Imavik Kvalvaag

(student(er))

Avtalen angir avtalepartenes plikter vedrørende gjennomføring av prosjektet og rettigheter til anvendelse av de resultater som prosjektet frembringer:

1. Studenten(e) skal gjennomføre prosjektet i perioden fra 11.01.21 til 20.05.21.

   Studentene skal i denne perioden følge en oppsatt fremdriftsplan der NTNU IE på Gjøvik yter veiledning. Oppdragsgiver yter avtalt prosjektbistand til fastsatte tider. Oppdragsgiver stiller til rådighet kunnskap og materiale som er nødvendig for å få gjennomført prosjektet. Det forutsettes at de gitte problemstillinger det arbeides med er aktuelle og på et nivå tilpasset studentenes faglige kunnskaper. Oppdragsgiver plikter på forespørsel fra NTNU å gi en vurdering av prosjektet vederlagsfritt.

2. Kostnadene ved gjennomføringen av prosjektet dekkes på følgende måte:
   - Oppdragsgiver dekker selv gjennomføring av prosjektet når det gjelder f.eks. materiell, telefon, reiser og nødvendig overnatting på steder langt fra NTNU i Gjøvik. Studentene dekker utgifter for ferdigstillelse av prosjektmateriell.
   - Eiendomsretten til eventuell prototyp tilfaller den som har betalt komponenter og materiell mv. som er brukt til prototypen. Dersom det er nødvendig med større og/eller spesielle investeringer for å få gjennomført prosjektet, må det gjøres en egen avtale mellom partene om eventuell kostnadsfordeling og eiendomsrett.

3. NTNU IE på Gjøvik står ikke som garantist for at det oppdragsgiver har bestilt fungerer etter hensikten, ei heller at prosjektet blir fullført. Prosjektet må anses som en eksamensrelatert oppgave som blir bedømt av intern og ekstern sensor. Likevel er det en forpliktelse for utøverne av prosjektet å fullføre dette til avtalte spesifikasjoner, funksjonsnivå og tider.

4. Alle bestått bacheloroppgaver som ikke er klausulert og hvor forfatteren(e) har gitt sitt samtykke til publisering, kan gjøres tilgjengelig via NTNUs institusjonelle arkiv NTNU Open.

Tilgjengeliggjøring i det åpne arkivet forutsetter avtale om delvis overdragelse av opphavsrett, se «avtale om publisering» (jfr Lov om opphavsrett). Oppdragsgiver og veileder godtar slik offentliggjøring når de signerer denne prosjektavtalen, og må evt. gi skriftlig melding til studenter og instituttleder/fagenhetsleder om de i løpet av prosjektet endrer syn på slik offentliggjøring.

Den totale besvarelsen med tegninger, modeller og apparatur så vel som programlisting, kildekode mv. som inngår som del av eller vedlegg til besvarelsen, kan vederlagsfritt benyttes til undervisnings- og forskningsformål. Besvarelsen, eller vedlegg til den, må ikke nyttes av NTNU til andre formål, og ikke overlates til utenforstående uten etter avtale med de øvrige parter i denne avtalen. Dette gjelder også firmaer hvor ansatte ved NTNU og/eller studenter har interesser.

5. Besvarelsens spesifikasjoner og resultat kan anvendes i oppdragsgivers egen virksomhet. Gjør studenten(e) i sin besvarelse, eller under arbeidet med den, en patentbar oppfinnelse, gjelder i forholdet mellom oppdragsgiver og student(er) bestemmelsene i Lov om retten til oppfinnelser av 17. april 1970, §§ 4-10.

6. Ut over den offentliggjøring som er nevnt i punkt 4 har studenten(e) ikke rett til å publisere sin besvarelse, det være seg helt eller delvis eller som del i annet arbeide, uten samtykke fra oppdragsgiver. Tilsvarende samtykke må foreligge i forholdet mellom student(er) og faglærer/veileder for det materialet som faglærer/veileder stiller til disposisjon.

7. Studenten(e) leverer oppgavebesvarelsen med vedlegg (pdf) i NTNUs elektroniske eksamenssystem. I tillegg leveres ett eksemplar til oppdragsgiver.

8. Denne avtalen utferdiges med ett eksemplar til hver av partene. På vegne av NTNU, IE er det instituttleder/faggruppeleder som godkjenner avtalen.

9. I det enkelte tilfelle kan det inngås egen avtale mellom oppdragsgiver, student(er) og NTNU som regulerer nærmere forhold vedrørende bl.a. eiendomsrett, videre bruk, konfidensialitet, kostnadsdekning og økonomisk utnyttelse av resultatene. Dersom oppdragsgiver og student(er) ønsker en videre eller ny avtale med oppdragsgiver, skjer dette uten NTNU som partner.

10. Når NTNU også opptrer som oppdragsgiver, trer NTNU inn i kontrakten både som utdanningsinstitusjon og som oppdragsgiver.

11. Eventuell uenighet vedrørende forståelse av denne avtale løses ved forhandlinger avtalepartene imellom. Dersom det ikke oppnås enighet, er partene enige om at tvisten løses av voldgift, etter bestemmelsene i tvistemålsloven av 13.8.1915 nr. 6, kapittel 32.

12. Deltakende personer ved prosjektgjennomføringen:

NTNUs veileder (navn): Marius Pedersen

Oppdragsgivers kontaktperson (navn): Knut Marius Myrvold

Student(er) (signatur): _____ dato 18/1 -21

_____ dato 18/1-21

_____ dato 18/1 - 21

_____ dato 18/1 - 21

Oppdragsgiver (signatur): _____ dato 18/1 - 21

Signert avtale leveres digitalt i Blackboard, rom for bacheloroppgaven.
Godkjennes digitalt av instituttleder/faggruppeleder.

Om papirversjon med signatur er ønskelig, må papirversjon leveres til instituttet i tillegg.
Plass for evt sign:

Instituttleder/faggruppeleder (signatur): _____ dato _____

Vedlegg B

Prosjektoppgave

Oppdragsgiver: Norsk Institutt for Naturforskning (NINA)

Kontaktperson NINA: Knut Marius Myrvold / knut.myrvold@nina.no / 920 64 963

Kontaktperson NTNU: Marius Pedersen / marius.pedersen@ntnu.no / 936 34 385

Tittel: Artsgjenkjenning av fisk i video

Videoopptak brukes i økende grad i overvåking av natur, både på land og i vann. En viktig årsak er at video er en passiv og skånsom metode som ikke er avhengig av fangst og håndtering av dyr. Norsk Institutt for Naturforskning (NINA) bruker i dag undervannskamera til å filme fisk i ulike miljøer og for ulike formål. Vi har allerede kommet et steg på veien med automatisk bildegjenkjenning i «rene miljøer» slik som fisketrapper i elver. Her er det ofte glatte betongvegger, og objekter som beveger seg mot denne bakgrunnen kan oppdages med høy presisjon.

En ny type overvåking har blitt prøvd ut i 2020, nemlig filming av komplekse fiskesamfunn i sesongmessige habitater. Når vannstanden stiger under vårflommen strømmer mange fiskearter inn i oversvømte områder. Disse miljøene byr på utfordringer for både biologer og ingeniører: Grumsete vann, store ansamlinger av fisk av ulike arter, bevegelige alger og plankton gjør at det er mye støy i bildet. Utfordringen er å skille relevant informasjon i et mylder av bevegelige objekter.

NINA har sikret ca 10TB med videomateriale (1,8 MB/s) i løpet av sommeren 2020. Vi ønsker et verktøy som automatisk kan gjenkjenne arter i videomaterialet under ulike lys- og siktforhold, og som kan loggføre antallet av de ulike artene i et gitt tidsrom.

Det er ønskelig med et brukergrensesnitt til verktøyet. Interesse for eller bakgrunn innen maskinsyn/bildebehandling er ønskelig. Utviklingsmiljø og konkrete oppgaver bestemmes i dialog med oppdragsgiver.

Oppgaven passer en gruppe på 2-3 personer.

Vedlegg C

Prosjektplan

Artsgjenkjenning av fisk
Prosjektplan

Birger Johan Nordølum Eirik Osland Lavik
Kristian André Dahl Haugen Tom-Ruben Traavik Kvalvaag

January 31, 2021

Innhold

Acronyms

MOTA Multiple Object Tracking Accuracy. 2

MOTP Multiple Object Tracking Precision. 2

NINA Norsk institutt for naturforskning. 2

Ordliste

Average Precision [1] $\int_0^1 p(r)dr$ der p er precision og r er recall for en gitt klassifisering . 1

Mean Average Precision [1] $\frac{\sum_{q=1}^{Q} AP(q)}{Q}$ der Q er antall klassifiseringer og AP er Average Precision. . 2

1 Mål og rammer

1.1 Bakgrunn

Som avsluttende vurdering for ingeniørstudenter ved NTNU Gjøvik gjennomføres det en bacheloroppgave. Der vi blandt annet skal lære å arbeide på en systematisk måte, gjennomføres et prosjekt med en reell problemstilling fra et reelt firma. For mange vil dette være slutten på studietiden, og den skal derfor brukes til å forbedre oss til arbeidslivet.

Norsk institutt for naturforskning (NINA)[1] benytter videoovervåking for å kunne passivt og skånsomt undersøke natur på land og i vann. Tidligere har NINA overvåket fisketrapper [2], og fått utviklet teknologier for å analysere dette mer eller mindre automatisk [3]. I 2020 utvidet oppdragsgiver overvåkning til å inkludere komplekse fiskesamfunn i sesongmessige habitater. Med dette har NINA til nå samlet inn 10 TB med videomateriell som må undersøkes. Forsøk på å analysere dette for hånd har vist seg å være en svært tidkrevende jobb. NINA ønsker seg en løsning som kan analysere videomateriellet for å finne ut når det er fisk, hvilken art og hvor mange. Se Vedlegg D for beskrivelse fra NINA. Dette prosjektet skal undersøke mulige løsninger for å automatisk gjenkjenne og telle fisk.

1.2 Prosjektmål

Oppgaven skal resultere i en løsning som skal kunne betydelig redusere tiden oppdragsgiver benytter for å analysere videofiler av fisk i sesongmessige habitater.

1.2.1 Resultatmål

Det skal produseres en løsning for å detektere, spore og artsbestemme fisk i en videostrøm. Gjennom et enkelt og intuitivt brukergrensesnitt skal brukeren kunne utføre disse. Løsningen skal returnere en rapport som viser antall fisk av bestemt art, tidspunkt den kom inn i videoen og tidspunkt ut. Denne rapporten skal være på et format som kan enkelt importeres inn i andre program for videre analyse.

Deteksjon/artsbestemming trenger høy presisjon. Dette er for å sikre at dersom en fisk vises i videoen, skal dens posisjon detekteres og klassifiseres. Målet er en Mean Average Precision, $mAP_{IoU=0.5} = 0.6$.

Sporing anses å være like viktig som deteksjon. Ved dårlig sporing kan den samme fisken telles flere ganger. Målet er for Multiple Object Tracking Accuracy (MOTA) blir 60. Multiple Object Tracking Precision (MOTP) settes til 75.

Mennesklig tid ønskes å være svært lav. I dag ligger den på ett minutt per minutt. Basert på øvrige mål skal dette reduseres til 5 minutt per 30 minutt video.

Mål for Mean Average Precision er basert på resultat fra en tidligere oppgave med tilsvarende problemstilling[4]. For MOTA og MOTP er det basert på resultat

[1] `https://nina.no`, besøkt 29.01.2021

fra tester utført på Multiple Object Tracking Benchmark[2].

1.2.2 Effektmål

- Frigjøre arbeidskapasitet hos oppdragsgiver.
- Kunne videreutvikles av andre enn de som var involvert i prosjektet.
- Løsningen skal kunne desentraliseres slik at den kan kjøre på flere maskiner eller lokasjoner.

1.3 Rammer

Tidsrammen til prosjektet er 11. Januar 2021 til 20. Mai 2021. Da skal løsning være ferdig og rapport klar til levering.
Teknologirammene er som følger:

- Løsningen må fungere på Windows.
- Løsningen skal ha et enkelt og intuitivt brukergrensesnitt.
- Løsningen skal fungere på en selvstendig arbeidsstasjon.
- Løsningen skal detektere følgende arter:
  Abbor, Brasme, Gjedde, Gullbust, Mort, Rumpetroll, Stingsild, Vederbuk, Ørekyt

1.4 Ressursbehov

Det er behov for ekstra ressurser fra NTNU for maskinlæringsspesifikke prosesseringsjobber. Det er også behov for 10 TB lagring for å avlaste eksterne harddisker med rå- og treningsdata fra oppdragsgiver. Dette vil være med på å gjøre utviklingsprosessen mer effektiv da alle gruppemedlem har tilgang til like ressurser.

2 Omfang

2.1 Fagområde

Oppdragsgiver har i dag noen undervannskameraer de benytter til å studere fisk ved strategiske områder i elver og vann. Deres løsning for å behandle denne dataen er å se gjennom flere dager med video i sanntid, for så å føre statistikk på fiskene underveis. Problemet med dette er at det tar veldig lang tid å se gjennom så mye video. Oppdragsgiver ønsker en automatisert løsning på dette som gjenkjenner hvilken fiskeart ut fra undervannsvideo.

På grunn av at undervannskamera ofte er plassert ute i felten, er det ikke alltid strøm eller internett tilgjengelig. Kameraene bruker strømnettet om det er tilgjengelig, dersom ikke, blir det brukt batteri eller solcelle. Selve videofilene blir tatt opp og lagret på en harddisk ute i felten. Etter en tid må diskene tømmes manuellt og analyseres.

[2] `https://motchallenge.net/`, besøkt 20.01.21

Statistikken oppdragsgiver skaper bidrar til å undersøke hvilke fisk som bruker viktige gyteområder. Noe som kan bidra til at viktige naturlige områder for fiskebestander blir bevart.

2.2 Oppgavebeskrivelse

Den optimale løsningen gruppen ser for seg er et verktøy som gjør at oppdragsgiver sine forskere kan lettere studere fisk. Dersom løsningen kan ta seg av det mest tidskrevende med gjennomgang av videomateriellet, vil oppdragsgiver kunne ha mer tid andre prosjekter. Samtidig vil oppdragsgiver om ønskelig kunne utvide bruk av videoovervåking, som kan bidra til flere forskningsprosjekter.

Siden løsningen skal kunne brukes av forskere, må den være enkel og intuitiv å bruke. På grunn av dette ser gruppen bort fra en kommandolinjeløsning, da ikke alle er komfortabel med å bruke et slikt verktøy. Derfor må det lages et grafisk brukergrensesnitt.

Hovedfokuset til oppgaven blir på maskinsyn og kunstig intelligens for å kunne detektere fisk. Dette vil innebære å opprette en modell for å kunne gjenkjenne og artsbestemme ut fra videomateriellet.

Slik som gruppen ser for seg løsningen vil oppdragsgiver tilkoble harddisker med video til en arbeidstasjon. Løsningen oppdager at disk med video er tilkoblet. Forskeren kan deretter navigere til brukergrensesnittet for å starte en prosesseringsjobb. Forskeren vil kunne velge en mappe på arbeidstasjon, som deretter vil brukes som et utgangspunkt for en batch-prosseseringsjobb. Jobben vil gå gjennom valgt videometeriell automatisk. Denne vil kunne oppdage, artsgjenkjenne og spore fisk på videostrømmen.

Resultatet av en prosesseringsjobb vil bli lagret fortløpende til en intern database for å holde på statistikken som blir generert. Løsningen vil fra databasen kunne gi fleksibilitet til å representere dataene slik som oppdragsgiver ønsker. Statistikk fra flere jobber kan kombineres om det gir logisk mening for forskningsarbeidet. Noe som gjør det enda mer intuitivt å hente ut data til forskning. For eksempel kan dette være data fra samme elv eller fysiske plassering.

Resultatet av analysert video vil bli gjort tilgengelig gjennom brukergrensesnittet. Noe som gjør at oppdragsgiver sine forskere kan kartlegge når det er aktivitet, hvilke arter fisk som er aktive og annen statistikk som er av interesse. Rådata vil også bli kunne eksportert til en fil som kan videre brukes i andre applikasjoner, til eksempel regneark.

2.3 Avgrensning

Bruker- og tjenerautentisering blir ikke del av bacheloroppgaven. Brukergrensesnittet blir gjort tilgjengelig på arbeidstasjonen hvor løsningen kjører, eller lokalt på nettverket om det kjører på en server. Dersom uvedkommende får tilgang er det ikke annet å se enn data fra tidligere jobber eller starte og stoppe prosesseringsjobber.

På grunn av mengden treningsdata fra oppdragsgiver er det ikke alle arter som det er nok videomateriell av til å skape et godt treningsett. Derfor avgrenses

oppgaven til å artsgjenkjenne fisk som vi har nok videomateriell til. For arter vi har utilstrekkelig data for, vil det samles til en generell samlekategori, som kan manuelt kategoriseres av forskere senere.

3 Prosjektorganisering

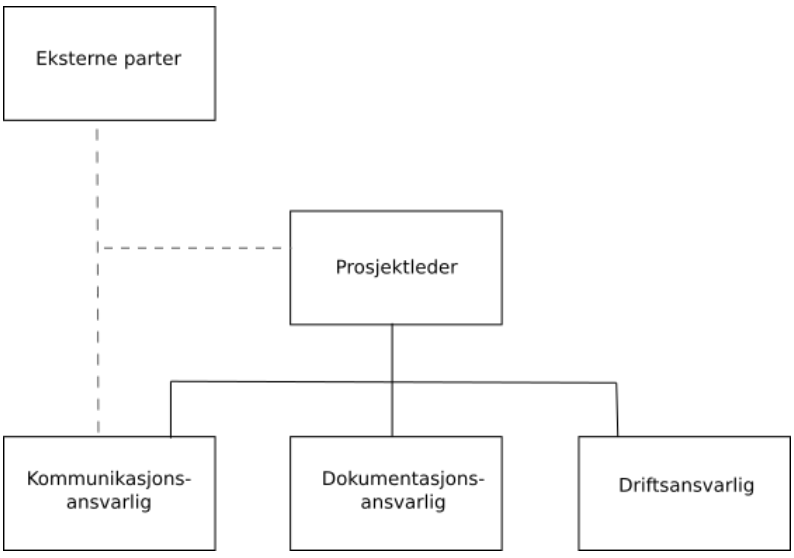
Prosjektgruppen består av fire dataingeniørstudenter. For å best mulig organisere arbeidet i gruppen er det definert gitte roller og ansvarsområder som definert i Avsnitt 3.1.

På grunn av smittesituasjonen i forbindelse med *COVID-19* planlegges det å gjennomføre prosjektet digitalt. Eventuelt på campus etter avtale internt i gruppen, om dette er tillatt. Dette kan endres underveis dersom nye retningslinjer fra myndighetene og NTNU kunngjøres.

3.1 Ansvarsforhold og roller

Prosjektroller

Internt i prosjektet er det definert noen prosjektroller med tilhørende ansvarsområder. Dette er gjort for å sikre at viktige elementer i prosjektarbeidet blir fulgt opp. Disse er definert i listen under og Figur 1.



Figur 1: Organisasjonskart for internt i gruppe

- **Prosjektleder** - Tom-Ruben T. Kvalvaag
  - Holde overordnet oversikt til prosjektet og styre arbeid i riktig retning som avtalt.
  - Siste ord ved avstemming uten flertall.
  - Ordstyrer i møter.

- **Dokumentansvarlig** - Kristian A. D. Haugen
  - Passe på at møter blir dokumentert og fulgt opp.
  - Oversikt av dokumentasjon i kode og organisere utvikling av bruker guide.
- **Kommunikasjonsansvarlig** - Birger J. Nordølum
  - Ansvarlig for kommunikasjonen ut fra og inn til gruppa.
- **Driftsansvarlig** - Eirik O. Lavik
  - Ansvar for å drifte infrastrukturen rundt miljøet som brukes i prosjektet.

I tillegg til internroller, er hvert medlem å betrakte som en utvikler. Alle gruppemedlem vil bidra til utviklingen ut fra valgt utviklingmodell i Avsnitt 4.1.1 om valg av utviklingsmetode.

Faglige ansvarsområder

Som en del av kvalitetssikringen skal hvert gruppemedlem ha sitt faglige ansvarsområde de fordyper seg ekstra innen. Den ansvarlige undersøker hva som er *state of the art* av forskning og teknologier innen fagfeltet.
Den ansvarlige innen hvert fagområde vil under prosjektet

- vurdere arbeidet gjort under sitt ansvarsområde
- svare på spørsmål
- holde intern kursing

Ut fra de områdene som sees på som kritiske for løsningen har følgende fagområder blitt valgt

- **Brukergrensesnitt og brukeropplevelse** - Birger J. Nordølum
- **Maskinlæring**
  - **Deteksjon** - Tom-Ruben T. Kvalvaag
  - **Sporing** - Eirik O. Lavik
- **Batchprosessering** - Kristian A. D. Haugen

3.2 Rutiner og regler i gruppen

Ett sett med grupperegler er utarbeidet som styrer interne rutiner og regler for gruppa, se Vedlegg B. Disse skal passe på at gruppen jobber effektivt og gi retningslinjer for hvordan gruppen skal håndtere problemer om det skal oppstå. Reglene kan revideres ved enighet blant alle medlemmene i gruppa.

4 PLANLEGGING, OPPFØLGING OG RAPPORTERING

4.1 Systemutviklingsmodell

For å effektivt gjennomføre et slikt prosjekt trengs det en utviklingsmodell [5]. Dette avgjør hvordan arbeidsflyten til gruppen skal være med å spesifisere hvordan arbeidsoppgaver skal fordeles, hvor lenge en arbeidsoppgave skal vare og arbeidsmetodikk mellom utviklerene.

4.1.1 Valg av systemutviklingsmodell

Ved bruk av en plandrevet modell er det vikig å vite nøyaktig hva oppdragssgiver ønsker. Dette betyr at oppdragsgiver og oppdragstaker må vere enige om løsningens funksjonalitet og virkemåte i starten av prosjektet. Fordelen her er at oppdragstaker kan arbeide effektivt uten involvering fra oppdragsgiver, om gjort korrekt. Det er derimot ikke like enkelt å gjøre endringer på planen underveis.

Alternativt, ved bruk av en smidig modell vil det være åpent for å gjøre større endringer ofterer. Kommunikasjon mellom oppdragsgiver og -taker er oftere og svært viktig. Det blir utført lite planlegging i starten til fordel for å planlegge underveis, sammen med oppdragsgiver og oppdragstaker.

Gruppen har ikke fått en ferdig kravspesifikasjon fra oppdragsgiver. Det er også vanskelig å estimere tid da merparten av gruppen mangler formell kompetanse for å utvikle kjernefunksjonalitet. På grunn av disse usikkerhetene er det vanskelig å estimere tid og funksjonalitet på en slik måte som en plandrevet utvikling krever. Det anntas også at en dynamisk utvikling vil gjøre det enklere for oppdragsgiver å forstå problemstillinger, samt bestemme seg for hva de ønsker.

Kanban[6] er en svært fleksibel smidig utviklingsmodell. Det sees store fordeler i hvordan en arbeidsoppgave beveges gjennom Kanban-brettet, og en relativ lav terskel i å hente en arbeidsoppgave. Gruppen er mer kritisk til friheten, og usikkerheter rundt det å ferdigstille en arbeidsoppgave, ettersom Kanban ikke har noen regler for når en påbegynt oppgave skal vere ferdig.

Scrum[7] løser problematikken med tidsfrister som finnes med Kanban. Sprinter sees på som en stor fordel for gruppen, da arbeidsoppgaver må ferdigstilles i løpet av en gitt tidsperiode. Det er ikke like ønskelig med det administrative rundt Scrum. Det er mange møter og en del formaliteter rundt produkteier, Scrummaster og gruppens utviklere. Det er derimot ønskelig med daglige møter der gruppen orientere hverandre på dems progressjon og status.

4.2 Anvending

Gruppen har endt med et kompromi av Scrum og Kanban, *Scrumban*. Gruppen velger å benytte Sprinter — herfra kallet iterasjoner — og daglige møter fra Scrum, samt Kanban-brett og *issue*-håndteringssystemet fra Kanban. Det er også ønskelig å ikke begrense arbeidsmengden til det bestemt under planleggingsmøtet, men å sette opp en prioriteringsliste over oppgaver som skal gjennomføres den gitte iterasjonen.

Kanban-brett

Brettet skal settes opp med følgende felt:

- bakkatalog
- ikke-påbegynt
- under arbeid
- til godkjenning
- ferdig

ikke-påbegynt vil være de høyest prioriterte arbeidsoppgaver den aktuelle iterasjonen, og skal fullføres i løpet av iterasjonen. *ferdig*-feltet skal tømmes etter endt iterasjon.

Iterasjoner

Iterasjoner skal starte på onsdager og vare i 10 arbeidsdager. Første iterasjon skal vare 6 dager, denne er forbeholdt initielle arbeidsoppgaver før koding starter. Siste iterasjon avsluttes på en fredag, og er da 2 dager lengre.

Statusmøter

Hver arbeidsdag kl. 09:30 skal det gjennomføres et statusmøte. Disse skal vare i opp til 20 minutt. Det skal gjennomgås hvert medlems arbeidsinnsats fra forrige dag, samt hva de skal arbeide med den dagen. Utover dette skal det arrangeres et iterasjonplanleggingsmøte på starten av hver iterasjon. Her skal arbeidsoppgaver prioriteres.

Det planlegges fra gruppen å arrangere faste møter med oppdragsgiver etter endt iterasjon. Her vil oppdragsgiver vises resultatet av iterasjonen. Gjennom møtet kan oppdragsgiver komme med tilbakemelding på løsningen. Tilbakemeldingen brukes i prioriteringsfasen for neste iterasjon.

4.3 Veiledningsmøte

Hver tirsdag kl. 09:00 til 10:00 skal gruppen ha møte med veileder for oppgaven. Her skal det gjennomgås hvordan gruppen ligger an i henhold til bacheloroppgaven.

5 Organisering av kvalitetssikring

5.1 Dokumentasjon, standardbruk og kildekode

Gruppen og oppdragsgiver er interessert i å publisere ferdig løsning som åpen kildekode. På grunn av dette er det viktig med gode rutiner på dokumentasjon og standardbruk. Implementasjonen av løsningen vil ha større fokus på kvalitet, enn om den var kun ment for én bruker (person, bedrift, gruppe). Dette er ikke samme som å si at koden ville ha en lavere kvalitet, men at apparatet rundt selve koden får en større rolle. I tillegg er et ekstra mål at løsning skal bli videreutviklet

av senere bachelorgrupper eller av eksterne bidragsytere. Noe som innebærer at koden vil være fritt tilgjengelig for alle til å bidra. Grunnet dette må koden være av god kvalitet og godt dokumentert.

For selve deteksjonene, benyttes resultatmålet fra Avsnitt 1.2.1 som referansen for om målet oppnås og dermed å kunne konkludere om kvaliteten er ivaretatt.

Dokumentasjon

Det er særdeles viktig at gjennom prosjektet at det er stor fokus på god dokumentasjon. Siden implementasjonen skal potensielt utvikles videre og brukes av mennesker som ikke nødvendigvis er like datakyndig som utvikler. Her er det viktig at dokumentasjonen følger utviklingen. Både at den er relevant og holdes opppdatert.

Standarder

For å kvalitetssikre løsningen vil velkjente standarder brukes for å sikre at koden følger en viss mal. Under har vi listet de standardene som er aktuelt for løsningen. De fleste av de gjelder for koden, men også noen rundt utformingen av brukergrensesnittet. Universell utforming er viktig i design av brukergrensesnittet.

Python

- Style Guide for Python Code - PEP8 [8]
- Type Hints - PEP 484 (Provisional) [9]
- Docstrings - PEP 257 [10]
    - *numpydoc*[3]
- Build-system - PEP 517 (Provisional) [11]

De som har *Provisional* i navnet er ikke endelige vedtatte standarder, men er annerkjent til å være tilnærmet som. Konsekvensen med at de ikke er endelige er at de kan forandre seg under eller etter prosjektet. Dette anses ikke som et problem for prosjektet.

numpydoc er ikke en offisiell standard, men er en god struktur å følge når dokumentasjon i koden skal skrives.

Brukergrensesnitt

- Nettlesarbaserte tenester - Digdir[4]
- *Web Content Accessibility Guidelines (WCAG)*[5]
- *Colour Contrast Analyser*[6]

[3] `https://numpydoc.readthedocs.io/en/latest/format.html`, besøkt 28.01.202

[4] `https://digdir.no/digitale-felleslosninger/nettlesarbaserte-tenester/1491`, besøkt 28.01.2021

[5] `https://w3.org/WAI/standards-guidelines/wcag/`, besøkt 28.01.2021

[6] `https://developer.paciellogroup.com/resources/contrastanalyser/`, besøkt 28.01.2021

Kodekvalitet

På grunnlag av store deler av oppgaven går ut på maskinlæring, og Python ofte er brukt som språket for disse systemene, var det naturlig å benytte samme språk i resten av implementasjoen. Foruten selve brukergrensesnittet, vil så mye som mulig skrives i Python. Siden ikke alle i gruppen har like god erfaring med språket, vil det benyttes ulike verktøy for å sikre god kodekvalitet.

For å ivareta kvalitet, vil det gjennom prosjektet benyttes en rekke verktøy. Disse kan hjelpe med formattering, sørge for at alt er dokumentert, følger en gitt standard, har god logikk, og liten kompleksitet. Det benyttes et par ulike løsninger som skal hjelpe med å sjekke for disse punktene. Ene er *pre-commit*[7] som vil sjekke koden før den dyttes til kodebrønnen og *GitLab pipelines*[8] som vil kjøre når koden er dyttet og det er laget en fletteanmodning. Mellom disse to, vil de aktuelle verktøyene bli satt opp litt ulikt. Blandt annet er det ikke logisk å kjøre kodekvalitet-sjekken lokalt, som kan ta lang tid. Denne er bedre om kjøres når koden er dyttet.

Noe av disse verktøyene er som følger, og disse prøver å følge standardene som nevnt i Avsnitt 5.1.

- *black*[9] (formatter): Kodestilen følger et bestemt sett regler. black følger PEP8 [8], og har i tillegg sine egne regler.
- *mypy*[10] (type checker): Sjekke om alle datatyper er eksplisitt definert vha. type definisjoner.
- *pyflakes*[11] / *pylint*[12] (kodeanalyse): Koden inneholder ikke feil.
- *coverage*[13] (tester): Passe på at koden blir sjekket av tester. Sjekke dekning av alle linjer i løsningen er ikke realistisk, men høy dekning er å foretrekke.

Testing

Gjennom valget av arkitektur er det svært nødvendig at det benyttes testing. Når disse modulene utvikles individuelt eller av kun et par gruppemedlem, er det viktig å sjekke at modulene kan kommunisere og ved eventuelle problemer, oppdage disse tidlig i utviklingen.

De flest kjente metodene for testing vil bli brukt på løsningen. Disse er akseptans-, bruker-, integrasjon- og unit-testing.

- *Akseptanse*: Testing utført av oppdragsgiver for å avgjøre om løsningen oppfyller krav før den overleveres.
- *Bruker*: Brukere hos oppdragsgiver vurderer løsningen ved bruk og kommer med tilbakemeldinger til utviklere.

[7] `https://pre-commit.com`, besøkt 29.01.2021
[8] `https://docs.gitlab.com/ce/ci/pipelines/`, besøkt 29.01.2021
[9] `https://black.readthedocs.io/en/stable/`, besøkt 28.01.2021
[10] `http://mypy-lang.org`. besøkt 28.01.2021
[11] `https://github.com/PyCQA/pyflakes`, besøkt 28.01.2021
[12] `https://pylint.org`, besøkt 28.01.2021
[13] `https://coverage.readthedocs.io/en/coverage-5.4/`, besøkt 28.01.2021

- *Integrasjon*: Tester at de forskjellige enhetene eller programmene i løsningen fungerer sammen som en helhet.
- *Unit*: Tester at hver enhet eller funksjon gir rett verdi basert på parametre.

Det legges tilrette for at oppdragsgiver skal fortløpende være med å se utviklingen. Løsningen skal brukes av personer som ikke er like teknisk som gruppemedlemmene, derfor må den testes gjennom bruk. Den tilbakemeldigen er viktig slik at løsning blir så enkel å ta bruk at den ikke skaper forvirring. Resultatet er at oppdragsgiver skal være brukeren, og all tilbakemelding de gir er med på skreddersy løsningen bedre til deres bruk.

5.2 Konfigurasjonsstyring

GitLab benyttes for versjonskontroll og oppfølging av aktuelle punkter og funksjonaliteter. Ved å benytte GitLab kan det meste relatert til utvikling være sentralisert på ett sted. For å holde orden på utviklingen og oversikt, brukes issues for ting som enten er direkte relatert til implementasjonen eller som en gjøreliste for punkter relatert til utviklingsprosessen.

Ved åpning av issue vil det benyttes en ferdiglaget mal om hvordan issues skal fylles ut. Dette er med på å sikre at hvert issue har en tydelig beskrivelse av hva den prøver å oppnå. Resultat av dette vil bety at alle kan bidra uten å må kontakte forfatter av issue for å forstå budskapet.

Gjennom prosjektet benyttes følgende rutine. Dette er med på å legge opp til en god praksis mellom medlemmene.

- *GitLab issue*[14], benyttes for alle oppgaver
- Ting som skal løses eller krever en diskusjon lages det et issue på
- Hvert issue skal inneholde en beskrivende forklaring hva det ønsker å oppnå.
- Hver *fletteanmodning* skal inneholde en forklarende problemstilling, forventet resultat og hva denne *merge request*'en prøver å løse.
- Alle issuer/fletteanmodning skal merkes med korrekt navnelapp
- Tilegnes en *Epic*[15] hvis nødvendig.
- Commits skal følge *conventional commits*[16].

Programvare

Tabell 1 gir en liten oversikt over de forskjellige programmene vi ser for oss å benytte gjennom prosjektet. Listen gjelder for ferdigløsninger og ikke teknologier vi skal ta i bruk når vi implementerer løsningen. Listen anses ikke som en endelig liste, men setter et utgangspunkt for prosessen videre.

[14] `https://docs.gitlab.com/ce/user/project/issue_board.html`, besøkt 28.01.2021
[15] `https://docs.gitlab.com/ce/user/group/epics/`, besøkt 28.01.2021
[16] `https://conventionalcommits.org`, besøkt 28.01.2021

Tabell 1: Programvareliste

Navn	Beskrivelse
GitLab	Utviklerplattform
SharePoint	Filehåndtering
Clockify	Tidstakning
Discord/zoom	Kommunikasjon
OpenStack	Infrastruktur i skyen
cvat	Annotering
pre-commit	Kodesjekk ved commiting

5.3 Risikoanalyse

Det er foretatt risikoanalyse av de punktene gruppen anser som aktuelle for rapporten. Boken Risikoanalyse[12] er benyttet som hjelp i å foreta en så god vurdering som mulig. Se Tabell 2 for hvordan vi regner ut risikoindeksen.

Vedlegg C viser risikoanalysen og Tabell 3 viser risikohåndteringen.

Tabell 2: Risikomatrise

Sannsynlighet / konsekvens	1.Svært lite sannsynlig	2.Lite sannsynlig	3.Sannsynlig	4.Ganske sannsynlig	5.Svært sannsynlig
5.Katastrofalt					
4.Svært stor					
3.Stor					
2.Middels					
1.Liten					

6 Plan for gjennomføring

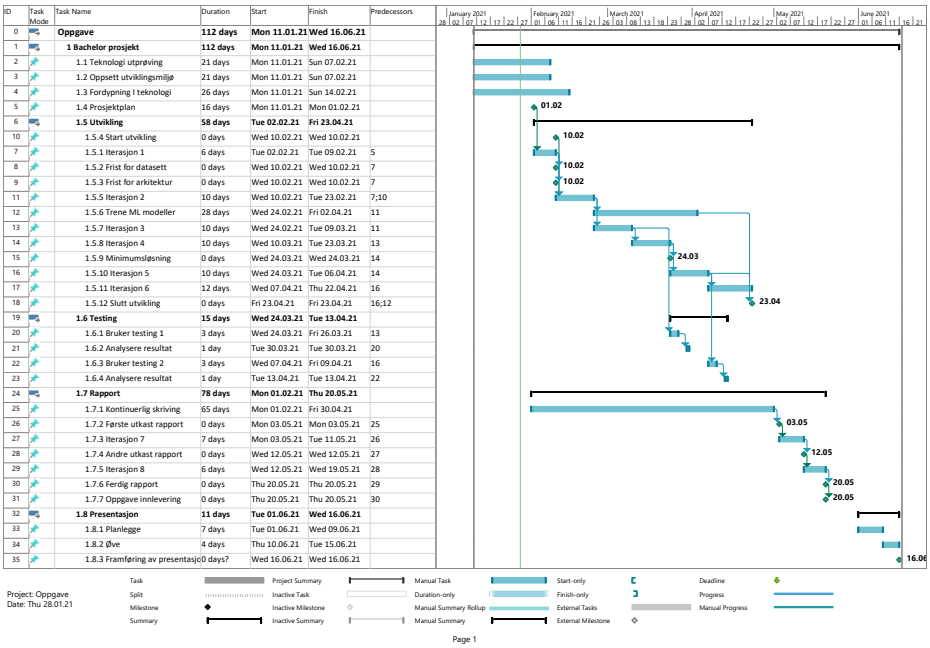
6.1 Tidsplan

En tidsplan over planlegging, utvikling, testing og rapportskriving er utarbeidet i et Gantt-diagram, se Figur 2.
Følgende milepæler er satt i prosjektet.

1. **Prosjektplan 01.02.2021:** Prosjektplan ferdig og levert til veileder.
2. **Datasett 10.02.2021:** Datasett ferdig annotert og klart for bruk.
3. **Arkitektur 10.02.2021:** Frist for valg av arkitektur og domenemodell skisse.
4. **Start utvikling 10.02.2021:** Start av første iterasjon med programmering.
5. **Minimums løsning ferdig 24.03.21:** En minimums løsning klar for brukertester av NINA.
6. **Brukerdokumentasjon utkast 24.03.21:** Utkast av bruker dokumentasjon for bruker testing.

Tabell 3: Risikotiltak

Nr.	Tiltak
1	Siden gruppemedlemmene ikke har mye erfaring, er det viktig av hvert medlem å være bevisst på han eller hennes begrensning, og i tillegg formidle denne gjennom prosjektet. Gjennom å være tydelig er det lettere å forutsi tidsbruk og resulterende funksjonalitet. Tiltak er å lage en god fremdriftsplan som er laget av alle i gruppen og som tydelig har angitte tidspunkter hvor funksjoner skal være ferdig. Ved avvik, kan det reflekteres over årsak og gjøre de nødvendige endringene for at avvik ikke skjer igjen.
2	Koden er i dag lastet opp til en GitLab-instans som tilbys av instituttet IDI[17]. Ved bruk av denne instansen er tilliten høy for at denne ikke går ned. Sannsynligheten for dette er minimal, slik at ingen tiltak vurderes. Eventuell mitigering er å laste opp koden til en annen tilbyder i tillegg. Samtidig eksisterer koden lokalt hos hver utvikler da desentralisert versjonskontroll brukes.
4	Da målet er at løsningen skal utvikles videre, må det tas høyde for at den må være av en så god kvalitet, i det minste være oversiktlig, slik at nye utviklere er villig til å bidra ved en senere anledning. Prosjektet benytter en rekke standarder og verktøy for at løsningen skal være av en slik kvalitet at det ikke skal være en barriere å bidra. Disse er brukt i kjente prosjektet, slik at gruppen anser de som en god pekepinn i riktig retning.
7	Valget av modulbasert arkitektur krever at disse snakker sammen på en god og effektiv måte. Gruppen har lite erfaring med denne type arkitektur og som konsekvens kan dette vise seg å være utfordrende. Her er det viktig at gruppen som en helhet jobber sammen, og den som utvikler modulen passer på at den er laget på en slik måte at den lett kan tas i bruk. Samarbeid og dokumentasjon er viktig punkter. Det skal etableres faste grensesnitt av modulene som kun eksponerer de funksjonene som kan brukes av andre moduler.
8	Løsningen baser seg på å gå gjennom videomateriell for deteksjon av fisk. Taper gruppen denne dataen, mistes noe av grunnlaget for å utvikle løsningen. Derfor er det viktig at første kopien av dataen, eksterne disker utleverte fra oppdragsgiver, kun brukes til å lese fra. All form for deteksjon, manipulering foregår på en kopi.
9	En viktig del av oppgaven er å samle statistikk over når fisken er i bildet. For at denne statistikken skal være korrekt, spores den samme fisken mellom bildene. Denne algoritmen må være såpass god at hver fisk ikke telles flere ganger så lenge fisken ikke forlater bilde. Om denne ikke er god nok, vil det foregå overlappende deteksjon. Her settes det av tid for å sikre at sporingen følger kvaliteten av deteksjonene.



ID	Task Mode	Task Name	Duration	Start	Finish	Predecessors
0		**Oppgave**	**112 days**	**Mon 11.01.21**	**Wed 16.06.21**	
1		**1 Bachelor prosjekt**	**112 days**	**Mon 11.01.21**	**Wed 16.06.21**	
2		1.1 Teknologi utprøving	21 days	Mon 11.01.21	Sun 07.02.21	
3		1.2 Oppsett utviklingsmiljø	21 days	Mon 11.01.21	Sun 07.02.21	
4		1.3 Fordypning I teknologi	26 days	Mon 11.01.21	Sun 14.02.21	
5		1.4 Prosjektplan	16 days	Mon 11.01.21	Mon 01.02.21	
6		**1.5 Utvikling**	**58 days**	**Tue 02.02.21**	**Fri 23.04.21**	
10		1.5.4 Start utvikling	0 days	Wed 10.02.21	Wed 10.02.21	
7		1.5.1 Iterasjon 1	6 days	Tue 02.02.21	Tue 09.02.21	5
8		1.5.2 Frist for datasett	0 days	Wed 10.02.21	Wed 10.02.21	7
9		1.5.3 Frist for arkitektur	0 days	Wed 10.02.21	Wed 10.02.21	7
11		1.5.5 Iterasjon 2	10 days	Wed 10.02.21	Tue 23.02.21	7;10
12		1.5.6 Trene ML modeller	28 days	Wed 24.02.21	Fri 02.04.21	11
13		1.5.7 Iterasjon 3	10 days	Wed 24.02.21	Tue 09.03.21	11
14		1.5.8 Iterasjon 4	10 days	Wed 10.03.21	Tue 23.03.21	13
15		1.5.9 Minimumsløsning	0 days	Wed 24.03.21	Wed 24.03.21	14
16		1.5.10 Iterasjon 5	10 days	Wed 24.03.21	Tue 06.04.21	14
17		1.5.11 Iterasjon 6	12 days	Wed 07.04.21	Thu 22.04.21	16
18		1.5.12 Slutt utvikling	0 days	Fri 23.04.21	Fri 23.04.21	16;12
19		**1.6 Testing**	**15 days**	**Wed 24.03.21**	**Tue 13.04.21**	
20		1.6.1 Bruker testing 1	3 days	Wed 24.03.21	Fri 26.03.21	13
21		1.6.2 Analysere resultat	1 day	Tue 30.03.21	Tue 30.03.21	20
22		1.6.3 Bruker testing 2	3 days	Wed 07.04.21	Fri 09.04.21	16
23		1.6.4 Analysere resultat	1 day	Tue 13.04.21	Tue 13.04.21	22
24		**1.7 Rapport**	**78 days**	**Mon 01.02.21**	**Thu 20.05.21**	
25		1.7.1 Kontinuerlig skriving	65 days	Mon 01.02.21	Fri 30.04.21	
26		1.7.2 Første utkast rapport	0 days	Mon 03.05.21	Mon 03.05.21	25
27		1.7.3 Iterasjon 7	7 days	Mon 03.05.21	Tue 11.05.21	26
28		1.7.4 Andre utkast rapport	0 days	Wed 12.05.21	Wed 12.05.21	27
29		1.7.5 Iterasjon 8	6 days	Wed 12.05.21	Wed 19.05.21	28
30		1.7.6 Ferdig rapport	0 days	Thu 20.05.21	Thu 20.05.21	29
31		1.7.7 Oppgave innlevering	0 days	Thu 20.05.21	Thu 20.05.21	30
32		**1.8 Presentasjon**	**11 days**	**Tue 01.06.21**	**Wed 16.06.21**	
33		1.8.1 Planlegge	7 days	Tue 01.06.21	Wed 09.06.21	
34		1.8.2 Øve	4 days	Thu 10.06.21	Tue 15.06.21	
35		1.8.3 Framføring av presentasjc	0 days?	Wed 16.06.21	Wed 16.06.21	

Figur 2: Gantt-diagram

7. **Utviklingslutt 23.04.21:** Siste iterasjon av utvikling ferdig.
8. **Første utkast rapport 30.04.21**
9. **Andre utkast rapport 30.04.21**
10. **Ferdig rapport 30.04.21**

6.2 Aktiviteter

For prosjektet er det funnet en oppdeling av aktiviteter som i listen under. Disse aktivitetene oppsummerer de arbeidsoppgavene som må fullføres for at prosjektets mål skal oppnås.

1. Artsgjenkjennings løsning
    1.1. Maskinlæring
        1.1.1. Datasett
        1.1.2. Lage modell
        1.1.3. Trene modell
        1.1.4. Evaluering
    1.2. Sporing
        1.2.1. Algoritme
        1.2.2. Evaluering
    1.3. Brukergrensesnitt

Bibliografi

[1] Wikipedia contributors, *Evaluation measures (information retrieval) — Wikipedia, The Free Encyclopedia*, https://en.wikipedia.org/w/index.php?title=Evaluation_measures_(information_retrieval)&oldid=1000641711, [Online; accessed 29-January-2021], 2021.

[2] T. H. Holter, K. M. Myrvold, U. Pulg og J. Museth, «Evaluating a fishway reconstruction amidst fluctuating abundances,» *River Research and Applications*, årg. 36, nr. 8, s. 1748–1753, 2020. DOI: https://doi.org/10.1002/rra.3688. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/rra.3688. adresse: https://onlinelibrary.wiley.com/doi/abs/10.1002/rra.3688.

[3] S. A. Nørstebø og E. Myrum, *Automatisk gjenkjenning av settefisk*, nor, 2019. adresse: http://hdl.handle.net/11250/2617894.

[4] H. A. W. Haugen, *Bruk av maskinsyn for automatisk telling og artsbestemmelse av villfisk som beiter under oppdrettsmerder*, 2020. adresse: https://hdl.handle.net/11250/2664006.

[5] I. Sommerville, *Software engineering*, eng, Boston Mass., 2016.

[6] W. contributors, *Kanban — Wikipedia, The Free Encyclopedia*, Online; accessed 28-January-2021, 2020. adresse: https://en.wikipedia.org/w/index.php?title=Kanban&oldid=996121229.

[7] W. contributors, *Scrum (software development) — Wikipedia, The Free Encyclopedia*, Online; accessed 28-January-2021, 2021. adresse: https://en.wikipedia.org/w/index.php?title=Scrum_(software_development)&oldid=1%20003155783.

[8] G. van Rossum, B. Warsaw og N. Coghlan, «Style Guide for Python Code,» PEP 8, 2001. adresse: https://www.python.org/dev/peps/pep-0008/.

[9] G. van Rossum, J. Lehtosalo og. Langa, «Type Hints,» PEP 484, 2014. adresse: https://www.python.org/dev/peps/pep-0484/.

[10] D. Goodger og G. van Rossum, «Docstring Conventions,» PEP 257, 2001. adresse: https://www.python.org/dev/peps/pep-0257/.

[11] N. J. Smith og T. Kluyver, «A build-system independent format for source trees,» PEP 517, 2015. adresse: https://www.python.org/dev/peps/pep-0517/.

[12] M. Rausand, *Risikoanalyse : teori og metoder*, nob, Trondheim, 2009.

A Proskjektavtale

NTNU

Vår dato Vår referanse

Norges teknisk-naturvitenskapelige universitet

Prosjektavtale

mellom NTNU Fakultet for informasjonsteknologi og elektroteknikk (IE) på Gjøvik (utdanningsinstitusjon), og

Norsk Institutt for Naturforskning (NINA)

(oppdragsgiver), og

Birger Johan Nordølum, Eirik Lavik, Kristian Haugen, Tom-Ruben Imavik Kvalvaag

(student(er))

Avtalen angir avtalepartenes plikter vedrørende gjennomføring av prosjektet og rettigheter til anvendelse av de resultater som prosjektet frembringer:

1. Studenten(e) skal gjennomføre prosjektet i perioden fra 11.01.21 til 20.05.21.

   Studentene skal i denne perioden følge en oppsatt fremdriftsplan der NTNU IE på Gjøvik yter veiledning. Oppdragsgiver yter avtalt prosjektbistand til fastsatte tider. Oppdragsgiver stiller til rådighet kunnskap og materiale som er nødvendig for å få gjennomført prosjektet. Det forutsettes at de gitte problemstillinger det arbeides med er aktuelle og på et nivå tilpasset studentenes faglige kunnskaper. Oppdragsgiver plikter på forespørsel fra NTNU å gi en vurdering av prosjektet vederlagsfritt.

2. Kostnadene ved gjennomføringen av prosjektet dekkes på følgende måte:
   - Oppdragsgiver dekker selv gjennomføring av prosjektet når det gjelder f.eks. materiell, telefon, reiser og nødvendig overnatting på steder langt fra NTNU i Gjøvik. Studentene dekker utgifter for ferdigstillelse av prosjektmateriell.
   - Eiendomsretten til eventuell prototyp tilfaller den som har betalt komponenter og materiell mv. som er brukt til prototypen. Dersom det er nødvendig med større og/eller spesielle investeringer for å få gjennomført prosjektet, må det gjøres en egen avtale mellom partene om eventuell kostnadsfordeling og eiendomsrett.

3. NTNU IE på Gjøvik står ikke som garantist for at det oppdragsgiver har bestilt fungerer etter hensikten, ei heller at prosjektet blir fullført. Prosjektet må anses som en eksamensrelatert oppgave som blir bedømt av intern og ekstern sensor. Likevel er det en forpliktelse for utøverne av prosjektet å fullføre dette til avtalte spesifikasjoner, funksjonsnivå og tider.

4. Alle bestått bacheloroppgaver som ikke er klausulert og hvor forfatteren(e) har gitt sitt samtykke til publisering, kan gjøres tilgjengelig via NTNUs institusjonelle arkiv NTNU Open.

Tilgjengeliggjøring i det åpne arkivet forutsetter avtale om delvis overdragelse av opphavsrett, se «avtale om publisering» (jfr Lov om opphavsrett). Oppdragsgiver og veileder godtar slik offentliggjøring når de signerer denne prosjektavtalen, og må evt. gi skriftlig melding til studenter og instituttleder/fagenhetsleder om de i løpet av prosjektet endrer syn på slik offentliggjøring.

Den totale besvarelsen med tegninger, modeller og apparatur så vel som programlisting, kildekode mv. som inngår som del av eller vedlegg til besvarelsen, kan vederlagsfritt benyttes til undervisnings- og forskningsformål. Besvarelsen, eller vedlegg til den, må ikke nyttes av NTNU til andre formål, og ikke overlates til utenforstående uten etter avtale med de øvrige parter i denne avtalen. Dette gjelder også firmaer hvor ansatte ved NTNU og/eller studenter har interesser.

5. Besvarelsens spesifikasjoner og resultat kan anvendes i oppdragsgivers egen virksomhet. Gjør studenten(e) i sin besvarelse, eller under arbeidet med den, en patentbar oppfinnelse, gjelder i forholdet mellom oppdragsgiver og student(er) bestemmelsene i Lov om retten til oppfinnelser av 17. april 1970, §§ 4-10.

6. Ut over den offentliggjøring som er nevnt i punkt 4 har studenten(e) ikke rett til å publisere sin besvarelse, det være seg helt eller delvis eller som del i annet arbeide, uten samtykke fra oppdragsgiver. Tilsvarende samtykke må foreligge i forholdet mellom student(er) og faglærer/veileder for det materialet som faglærer/veileder stiller til disposisjon.

7. Studenten(e) leverer oppgavebesvarelsen med vedlegg (pdf) i NTNUs elektroniske eksamenssystem. I tillegg leveres ett eksemplar til oppdragsgiver.

8. Denne avtalen utferdiges med ett eksemplar til hver av partene. På vegne av NTNU, IE er det instituttleder/faggruppeleder som godkjenner avtalen.

9. I det enkelte tilfelle kan det inngås egen avtale mellom oppdragsgiver, student(er) og NTNU som regulerer nærmere forhold vedrørende bl.a. eiendomsrett, videre bruk, konfidensialitet, kostnadsdekning og økonomisk utnyttelse av resultatene. Dersom oppdragsgiver og student(er) ønsker en videre eller ny avtale med oppdragsgiver, skjer dette uten NTNU som partner.

10. Når NTNU også opptrer som oppdragsgiver, trer NTNU inn i kontrakten både som utdanningsinstitusjon og som oppdragsgiver.

11. Eventuell uenighet vedrørende forståelse av denne avtale løses ved forhandlinger avtalepartene imellom. Dersom det ikke oppnås enighet, er partene enige om at tvisten løses av voldgift, etter bestemmelsene i tvistemålsloven av 13.8.1915 nr. 6, kapittel 32.

12. Deltakende personer ved prosjektgjennomføringen:

NTNUs veileder (navn): Marius Pedersen

Oppdragsgivers kontaktperson (navn): Knut Marius Myrvold

Student(er) (signatur): _____ dato 18/1 -21

_____ dato 18/1-21

_____ dato 18/1 - 21

_____ dato 18/1 - 21

Oppdragsgiver (signatur): _____ dato 18/1 - 21

Signert avtale leveres digitalt i Blackboard, rom for bacheloroppgaven.
Godkjennes digitalt av instituttleder/faggruppeleder.

Om papirversjon med signatur er ønskelig, må papirversjon leveres til instituttet i tillegg.
Plass for evt sign:

Instituttleder/faggruppeleder (signatur): _____ dato _____

B Grupperegler

Grupperegler

Gjelder:	Birger, Eirik, Kristian og Tom-Ruben		
Mål:	Gjøre en god bachelor. Ha et effektivt og godt samarbeid.		
Revisjon:	1	Dato:	25.01.2021

Arbeidsmetode

- Vi samarbeider om mål og planer for team-arbeidet.
- Vi avklarer enighet om mål og planer for hvert team-møte.
- Vi fordeler oppgaver likt, slik at mengden arbeid blir rettferdig.
- Vi holder gruppefrister.
- Vi jobber ut fra avtalte mål.
- Vi gir godt rom for drøfting av de temaene som tas opp.
- Vi gir alle mulighet til å bidra.
- Vi opprettholder kjerne arbeidstid fra 09:00 til 12:00, tirsdag til og med fredag, der vi er tilgjengelig for å diskutere problemer og hjelpe hverandre.
- Vi møtes tirsdag til fredag, kl. 09:30. Maks 20 min. Agenda; gjennomgang av hva som er gjort siste dag og plan videre fra hver enkelt.
- Vi planlegger å arbeide minst 30 timer i uken per person (600t/20uker).
- Vi fører timelister som ved slutten av hver uke er oppdatert.
- Vi tar avgjørelser der flertallet avgjør. Ved like mange voteringer på hvert alternativ diskuteres det en runde på maks. 10min for hvert alternative og ny votering tas. Om fremdeles ingen flertall, kan prosjektleder avgjøre votering med ekstra stemme.

Væremåte

- Jeg er ærlig med andre og meg selv.
- Jeg tar initiativ til oppgaver som må gjøres.
- Jeg gjennomfører individuelle oppgaver innen fastsatt frist.
- Jeg møter til avtalte tema- og gruppemøter.
- Jeg er fokusert på arbeidet når det jobbes.
- Jeg er tilgjengelig under gruppens kjernetid.
- Jeg gir beskjed om sykdom og andre hendelser som forhindrer oss i å jobbe som avtalt.
- Jeg ser løsninger og ikke fokuserer på begrensninger.
- Jeg holder ikke eierskap til deler av koden, og lar andre bidra.
- Jeg inkluderer alle i teamet og etterspør andres synspunkter og ideer.
- Jeg har respekt for andre og er åpen for ulike holdninger, meninger, og væremåter.
- Jeg tar opp grupperelaterte problemer på møter og ikke holder dem for oss selv.
- Jeg er positiv og medvirker til godt og humørfylt klima.
- Jeg oppmuntrer og motiverer andre når vi ser at noen er demotiverte.
- Jeg er lojal mot avgjørelser.

Konsekvens

1. Samtale i gruppen ved problem.
2. Skriftlig advarsel med kopi veileder ved gjentatte brudd på arbeidsmetode om flertallet er enig.
    a. Hvem det gjelder
    b. Hvilke brudd
    c. Tiltak for irettesetting
3. Samtale med veileder - etter to skriftlige advarsler.
4. Ekskludere fra gruppen etter enstemmig avstemming med alle andre i gruppen og veileder - etter flere samtaler med veileder.

Signaturer

Eirik Lavik _____ dato 25/01/21

_____ dato 25/01/21

Birger J. Nordahl _____ dato 25/01/21

Tom-R. V. Kvalvaag _____ dato 25/01/21

C Risikoanalyse

Nr.	Deloppgave	Fare/årsak	Mulige konsekvenser	Risiko			Risikoreduserende tiltak
				Sannsynlighet	Konsekvens	Risiko--index	
1	Produkt	Funksjonalitet overgår kunnskap og tid	Høy tidsforbruk Når ikke mål	4	3	7	Sett av tid for opplæring. Sett realistiske mål.
2		Tap av kodedepo	Hele prosjektet og implementasjonen blir satt tilbake till null	1	5	6	Bruke eksternt kodedepo (IDI sin GitLab). Dytte en backup et sted.
3		Tap av kode lokal på arbeidsstasjon	Utført arbeid som ikke er commited eller dyttet kan mistes.	3	1	4	Bruke versjonskontroll. Lage små commits og commite ofte.
4		Utvikle noe som kanskje er for komplisert i ettertid	Blir ikke utviklet videre	3	3	6	Følge standard. Bygge en god arkitektur.
5		Implementasjon ikke ferdig til deadline	Uferdig løsning som arbeidsgiver ikke kan bruke.	1	3	4	Prioritere minimumsløsning.
6		Lite datasett	Dårlig klassifisering og deteksjon	3	3	6	Annotere tilstrekklig med bilder. Bruke data agmetasjon under trening.
7		Kompleks moduldesign	Modulene snakker ikke sammen.	3	4	7	Planlegging av arkitektur. Benytte etablerte grensesnitt mellom moduler.
8		Tap av data (video) eller datasett	Mister materialet å utføre arbeidet på.	2	4	6	Ha en backup. Ikke jobb destruktivt på original dataen. Lagre datasett på nettverks disk som har redundanse
9		Manglende sporing	Teller feil antall fisk. Utfører feil statistikk.	3	3	6	Sett av god tid til å sørge for at sporingen fungerer bra. Se etter om eksisterende algoritmer kan forebedre resultat og har vist seg gode under vann.
10		Sky-basert utviklingsverktøy går ned (cvat, Windows vm)	Testing og arbeid settes tilbake	2	1	3	Sette opp lokalt vertøy for å utføre arbeidet.
11		Eksterne tjenesteleverandører midlertidig utilgjenlig	Forsinkelser eller avbrudd i arbeid	2	2	4	Rutine for å starte annet arbeid som kan gjøres lokalt for å ikke miste arbeidstid, som feks. skrive på rapport eller dokumentasjon.
12	Prosjekt	Sykdomsfravær	Arbeid hindres av at nøkkelpersonell kan sitte med kunnskap.	2	3	5	Alle setter seg litt inn i hver ting. God dokumentasjon. Opplæringen tvers av fagområde.
13		Uenighet i ønsket funksjonalitet eller implementasjon(design)	Tiden drar ut ved at enighet ikke fattes tidlig. Dårlig stemning innad i gruppen.	2	2	4	Sette opp tydlige regler som tar hånd om dette tidlig. Prosjektleder setter foten ned. Undersøke med oppgragsgiver om ønske.
14		Strengere COVID-19-tiltak	Hindrer muligheten for fysisk aktivitet.	3	2	5	Bruk maske og håndsprit. Følg råd gitt av styresmakter. Prioriter fjernarbeid.
15	Bedrift	Rapport ikke ferdig til deadline	Dårligere vurdering	1	4	5	Ha gode rutiner og fremdriftsplan som viser hvordan rapporten ligger an. Ukentlig møte med veileder. Kontinuerlig skriving i rapport.
16		For komplekst for bruker	Løsningen blir ikke brukt	3	2	5	Involver arbeidsgiver regelmessig. Bruk skisser og demoer tidlig. Planlegg bruk av brukertester underveis av prosjektet.
17		Dårlig modell for maskinlæring	Tap av rykte Ekstra arbeid for NINA Løsning blir ikke brukt	3	2	5	Godt treningssett. Legge tilrette for å videretrene modell

Vedlegg D

Hyperparameter

Standard og brukte hyperparameter innstillinger for *YOLOv5*.

```
# Hyperparameters for COCO training from scratch
# python train.py --batch 40 --cfg yolov5m.yaml --weights '' --data coco.
    yaml --img 640 --epochs 300
# See tutorials for hyperparameter evolution https://github.com/
    ultralytics/yolov5#tutorials


lr0: 0.01  # initial learning rate (SGD=1E-2, Adam=1E-3)
lrf: 0.2  # final OneCycleLR learning rate (lr0 * lrf)
momentum: 0.937  # SGD momentum/Adam beta1
weight_decay: 0.0005  # optimizer weight decay 5e-4
warmup_epochs: 3.0  # warmup epochs (fractions ok)
warmup_momentum: 0.8  # warmup initial momentum
warmup_bias_lr: 0.1  # warmup initial bias lr
box: 0.05  # box loss gain
cls: 0.5  # cls loss gain
cls_pw: 1.0  # cls BCELoss positive_weight
obj: 1.0  # obj loss gain (scale with pixels)
obj_pw: 1.0  # obj BCELoss positive_weight
iou_t: 0.20  # IoU training threshold
anchor_t: 4.0  # anchor-multiple threshold
# anchors: 3  # anchors per output layer (0 to ignore)
fl_gamma: 0.0  # focal loss gamma (efficientDet default gamma=1.5)
hsv_h: 0.015  # image HSV-Hue augmentation (fraction)
hsv_s: 0.7  # image HSV-Saturation augmentation (fraction)
hsv_v: 0.4  # image HSV-Value augmentation (fraction)
degrees: 0.0  # image rotation (+/- deg)
translate: 0.1  # image translation (+/- fraction)
scale: 0.5  # image scale (+/- gain)
shear: 0.0  # image shear (+/- deg)
perspective: 0.0  # image perspective (+/- fraction), range 0-0.001
flipud: 0.0  # image flip up-down (probability)
fliplr: 0.5  # image flip left-right (probability)
mosaic: 1.0  # image mosaic (probability)
mixup: 0.0  # image mixup (probability)
```

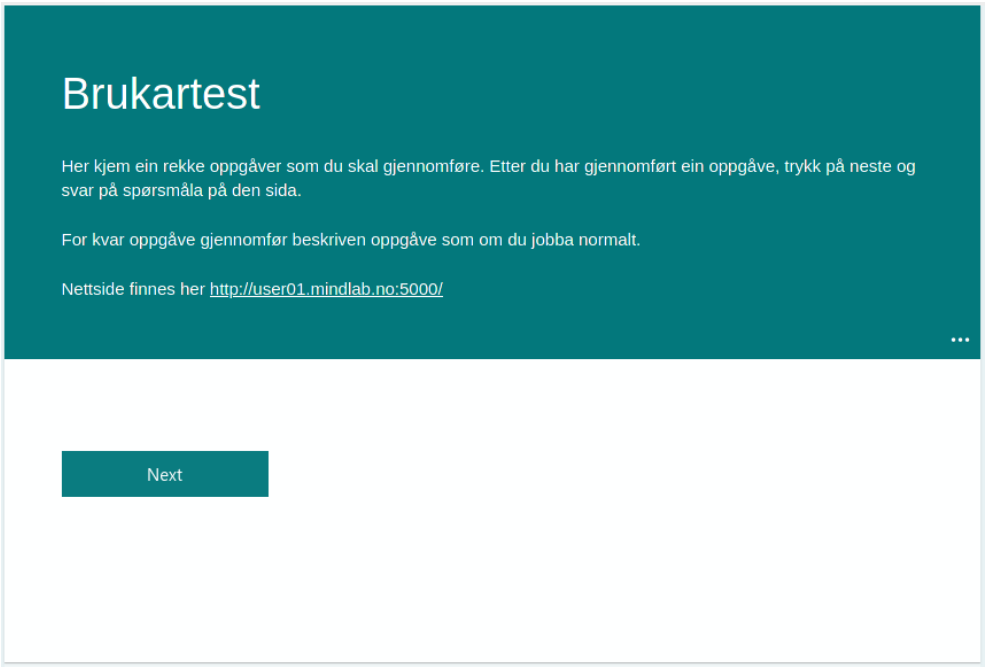
Kodeliste D.1: Standard hyperparameter innstillinger.

```
lr0: 0.01  # initial learning rate (SGD=1E-2, Adam=1E-3)
lrf: 0.2  # final OneCycleLR learning rate (lr0 * lrf)
momentum: 0.937  # SGD momentum/Adam beta1
weight_decay: 0.0005  # optimizer weight decay 5e-4
warmup_epochs: 3.0  # warmup epochs (fractions ok)
warmup_momentum: 0.8  # warmup initial momentum
warmup_bias_lr: 0.1  # warmup initial bias lr
box: 0.05  # box loss gain
cls: 0.5  # cls loss gain
cls_pw: 1.0  # cls BCELoss positive_weight
obj: 1.0  # obj loss gain (scale with pixels)
obj_pw: 1.0  # obj BCELoss positive_weight
iou_t: 0.20  # IoU training threshold
anchor_t: 4.0  # anchor-multiple threshold
# anchors: 3  # anchors per output layer (0 to ignore)
fl_gamma: 0.0  # focal loss gamma (efficientDet default gamma=1.5)
hsv_h: 0.2  # image HSV-Hue augmentation (fraction)
hsv_s: 0.7  # image HSV-Saturation augmentation (fraction)
hsv_v: 0.4  # image HSV-Value augmentation (fraction)
degrees: 45.0  # image rotation (+/- deg)
translate: 0.1  # image translation (+/- fraction)
scale: 0.75  # image scale (+/- gain)
shear: 0.0  # image shear (+/- deg)
perspective: 0.0  # image perspective (+/- fraction), range 0-0.001
flipud: 0.1  # image flip up-down (probability)
fliplr: 0.5  # image flip left-right (probability)
mosaic: 1.0  # image mosaic (probability)
mixup: 0.0  # image mixup (probability)
```

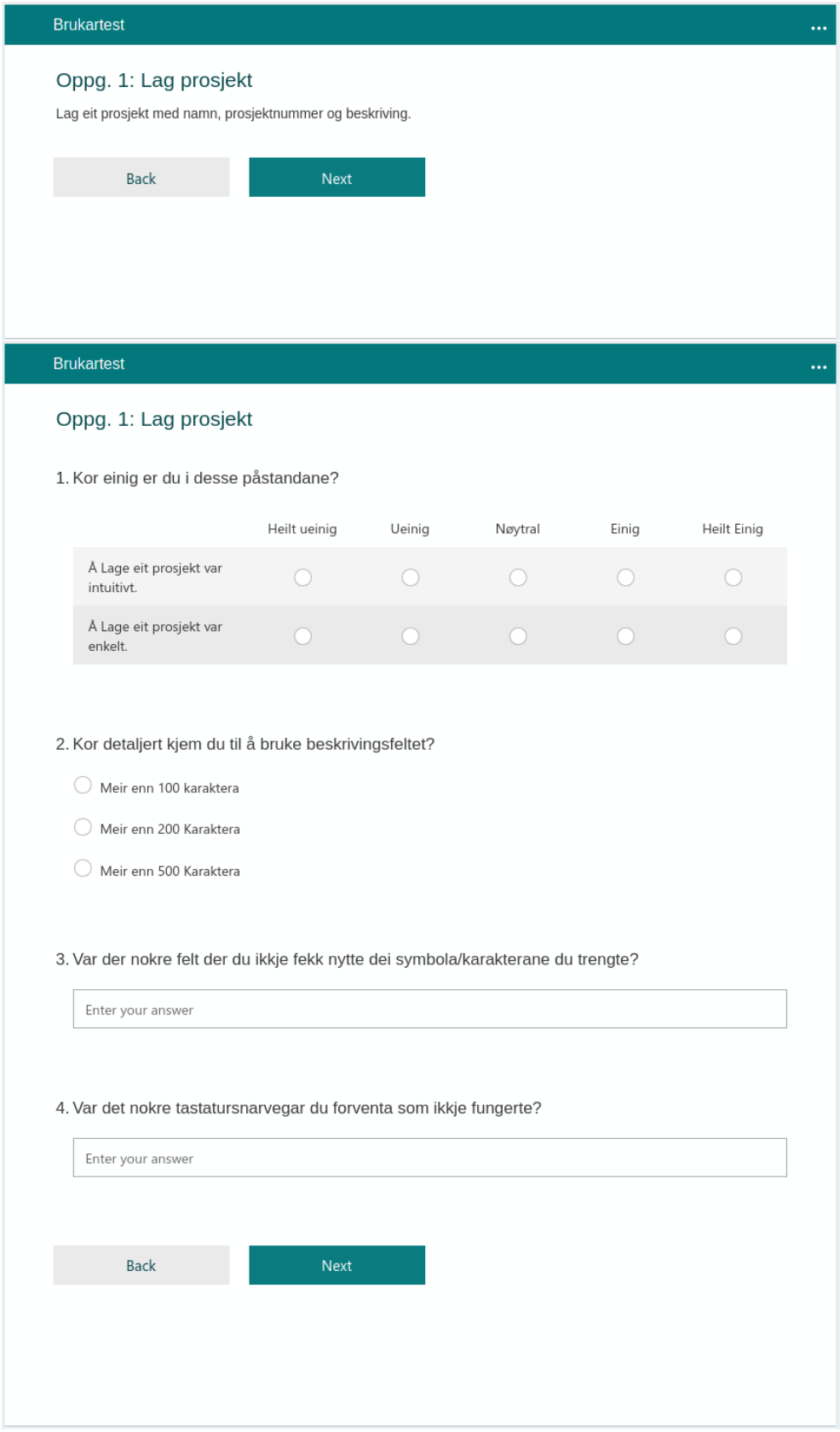
Kodeliste D.2: Brukte hyperparameter innstillinger.

Vedlegg E

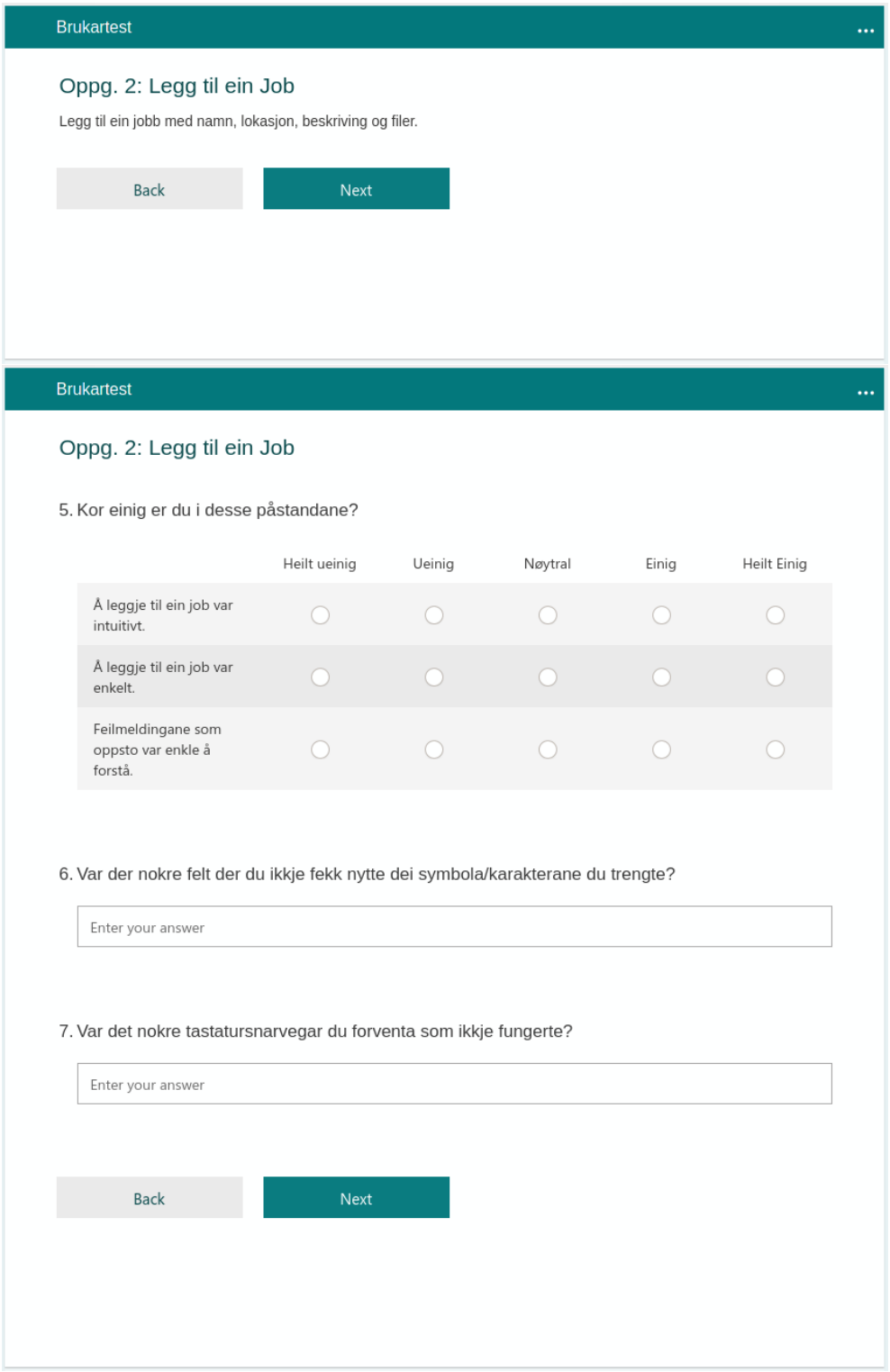
Brukertest skjema



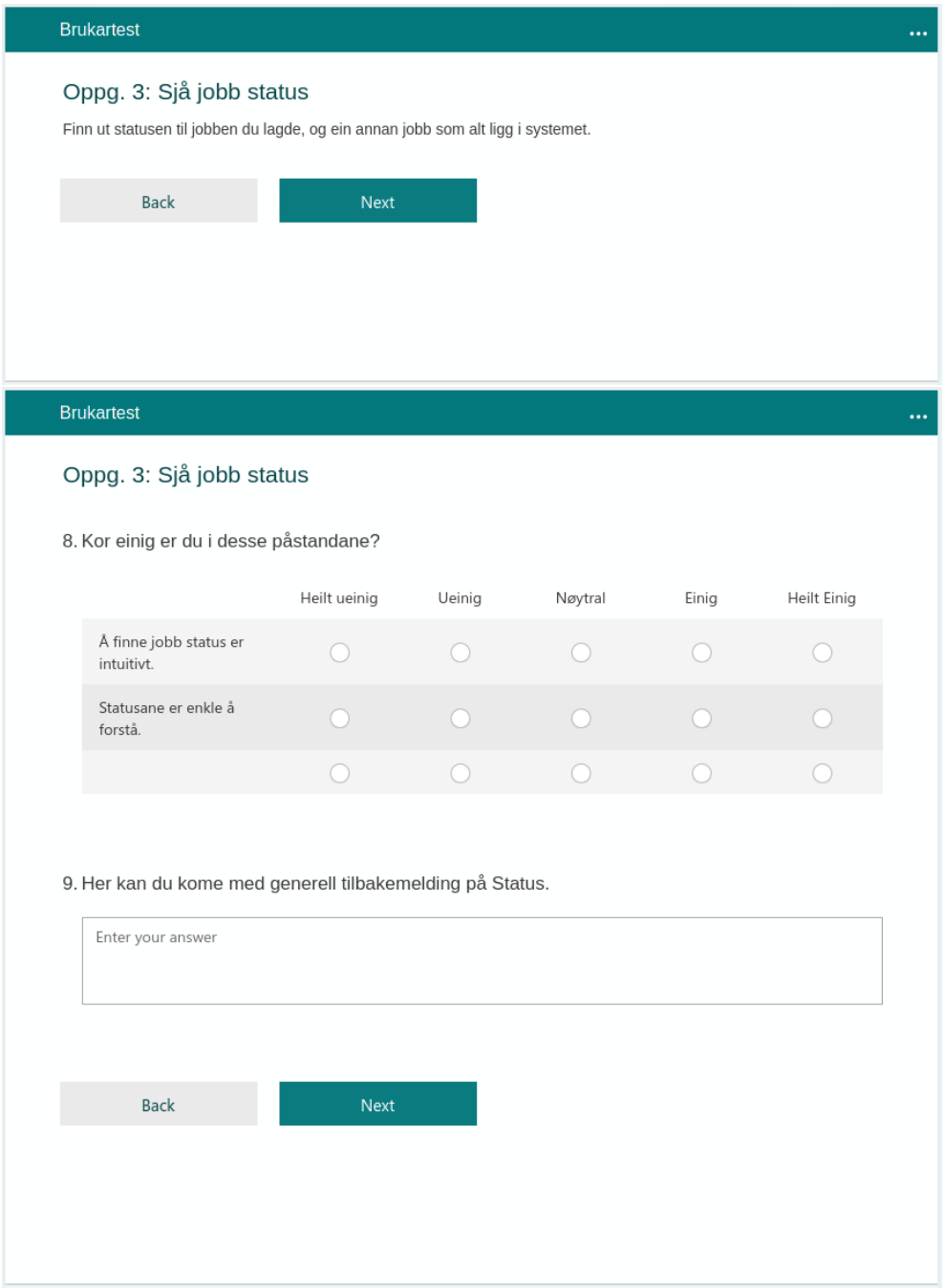
Figur E.1: Side 1 av 12



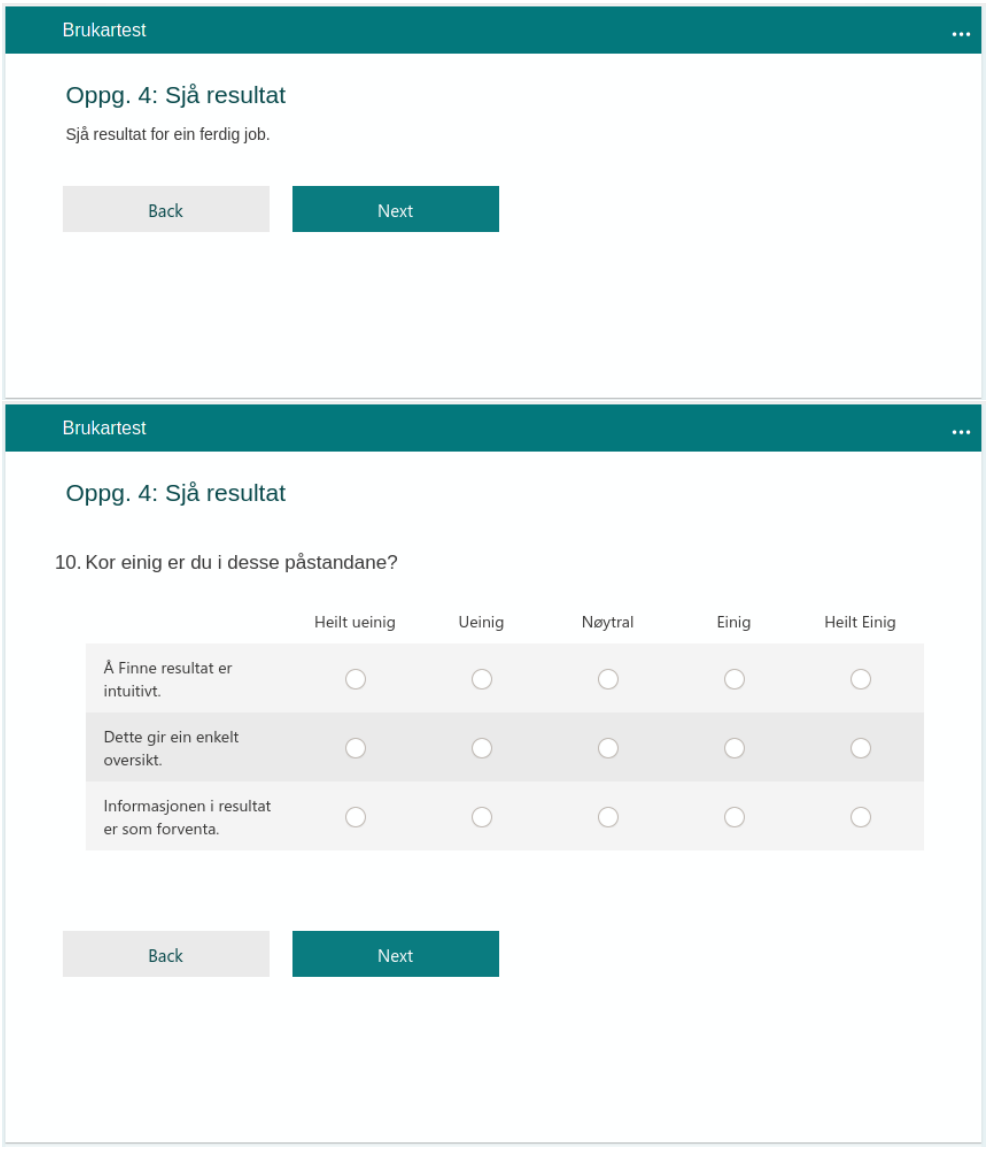
Figur E.1: Side 2 og 3 av 12



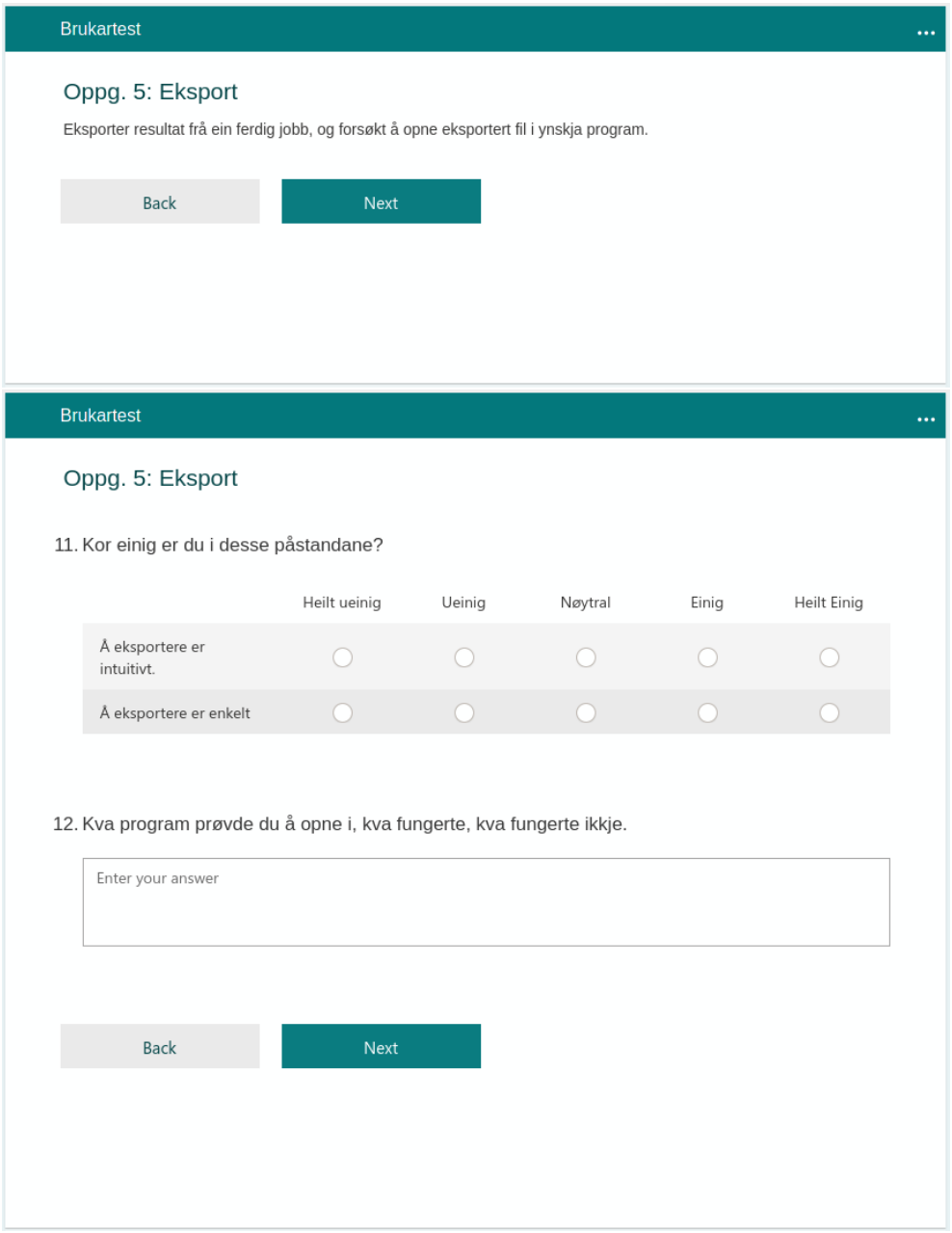
Figur E.1: Side 4 og 5 av 12



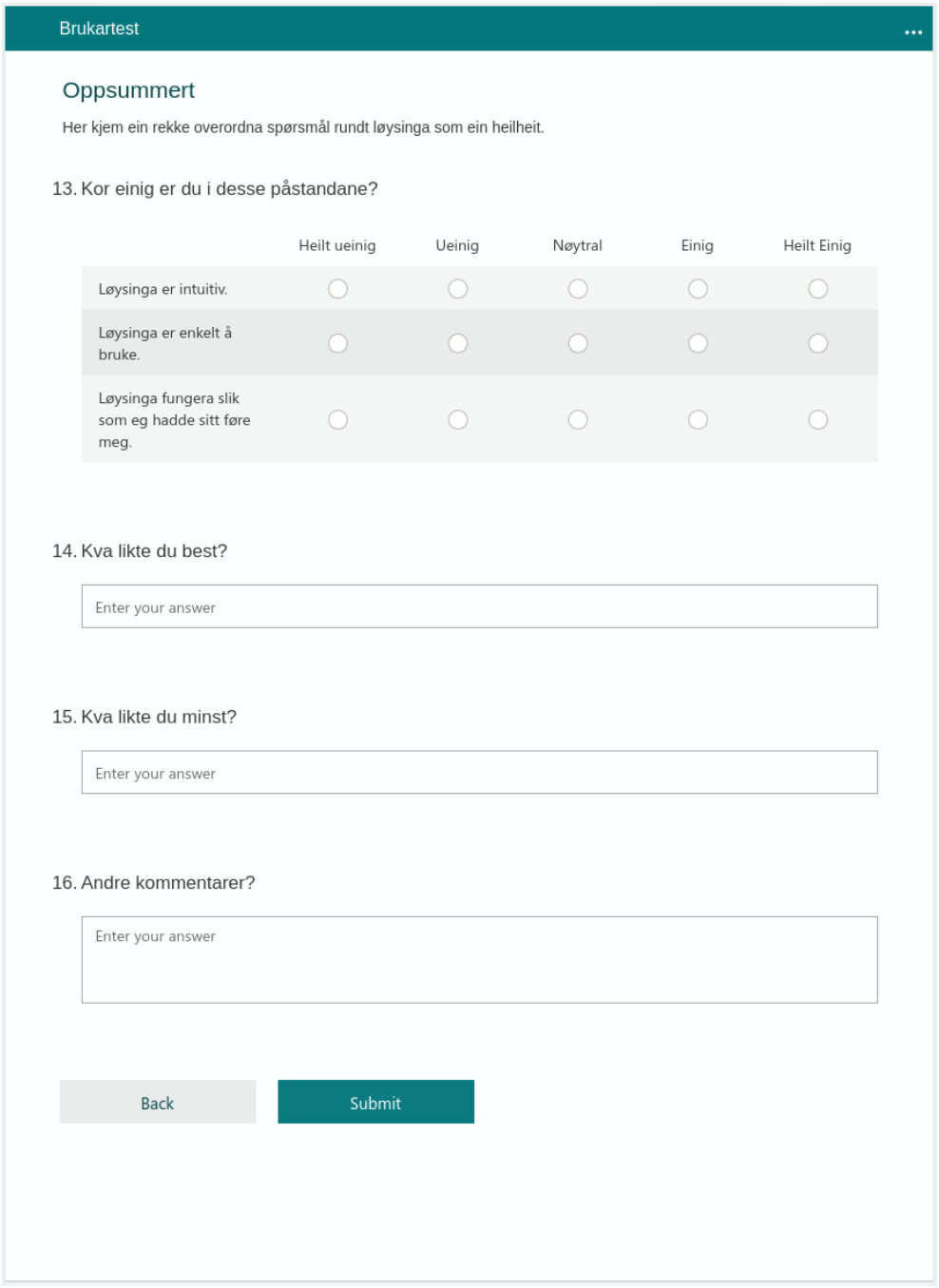
Figur E.1: Side 6 og 7 av 12



Figur E.1: Side 8 og 9 av 12



Figur E.1: Side 10 av 11



Figur E.1: Side 12 av 12

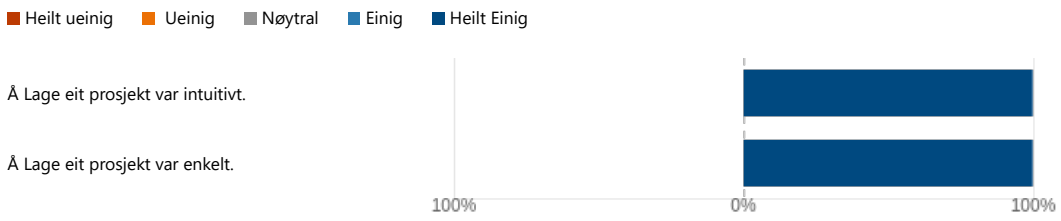
Vedlegg F

Brukertest resultat

Brukartest

2	31:27	Active
Responses	Average time to complete	Status

1. Kor einig er du i desse påstandane?



2. Kor detaljert kjem du til å bruke beskrivingsfeltet?

3. Var der nokre felt der du ikkje fekk nytte dei symbola/karakterane du trengte?

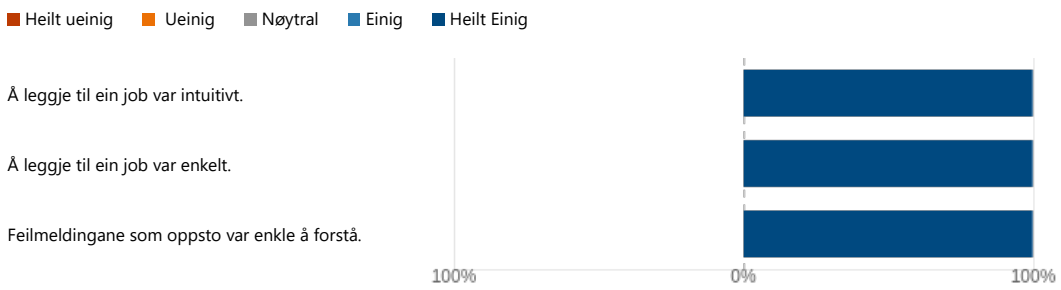
2	Latest Responses
Responses	*"Nei"*
	"Nei"

4. Var det nokre tastatursnarvegar du forventa som ikkje fungerte?

2
Responses

Latest Responses
"Prøvde ingen"
"Nei"

5. Kor einig er du i desse påstandane?



6. Var der nokre felt der du ikkje fekk nytte dei symbola/karakterane du trengte?

2
Responses

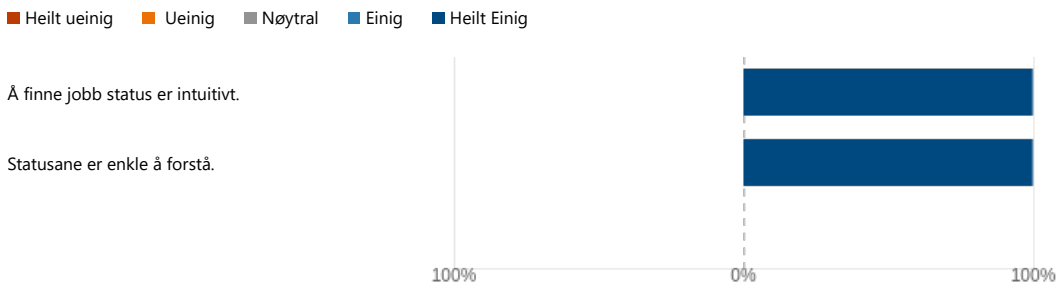
Latest Responses
"prøvde ikke"
"Nei"

7. Var det nokre tastatursnarvegar du forventa som ikkje fungerte?

2
Responses

Latest Responses
"prøvde ikke"
"Skulle muligens hatt mulighet til å gå tilbake å velge flere filer før ma…

8. Kor einig er du i desse påstandane?



9. Her kan du kome med generell tilbakemelding på Status.

2

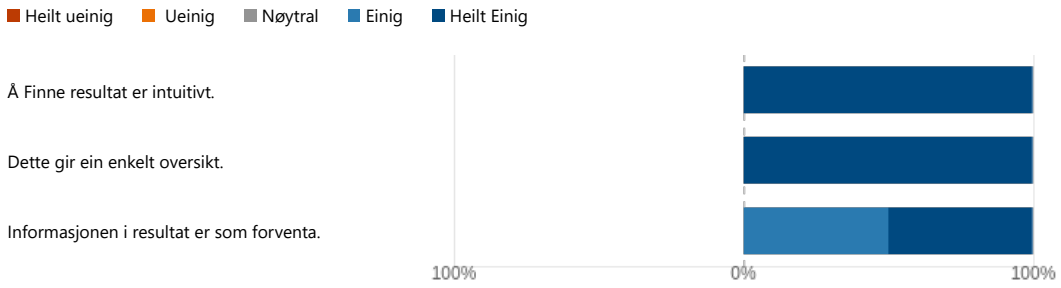
Responses

Latest Responses

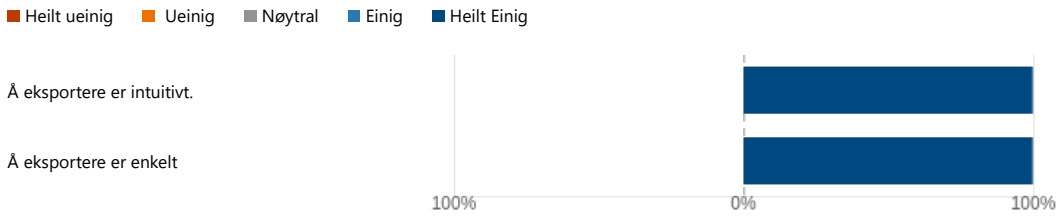
"Dette var veldig intuitivt. Lett å finne status for jobben, særlig at den s…

"Kunne vært hensiktsmessig med en knapp for å aktivt starte jobben. …

10. Kor einig er du i desse påstandane?



11. Kor einig er du i desse påstandane?



12. Kva program prøvde du å opne i, kva fungerte, kva fungerte ikkje.

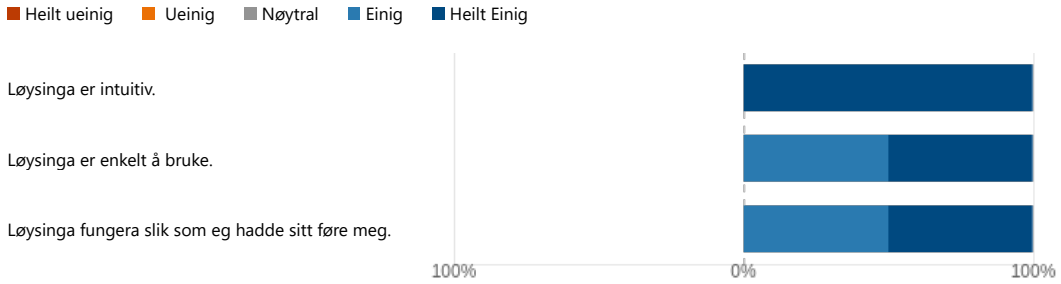
2
Responses

Latest Responses

"Excel - som forventet for denne typen filer (og til mitt bruk). Filformate…

"excel. Ikke separert ved åpning. Men dette er enkelt å gjøre manuelt.…

13. Kor einig er du i desse påstandane?



14. Kva likte du best?

2
Responses

Latest Responses

"Brukergrensesnittet er enkelt og intuitivt, og har en fin layout. Ingen u…

"Enkelt brukergrensersnitt, bra med felter for å presisere prosjektet. Fi…

15. Kva likte du minst?

2
Responses

Latest Responses

"Skjønte ikke hva video ID viste til i rapporten."

"savnet en mulighet til å kunne starte/pause jobben aktivt, evt også m…

16. Andre kommentarer?

2
Responses

Latest Responses

"Intuitivt og enkelt! Godt jobbet!"

"ID på resultatsiden bør samsvare med id i csv fil. Hvilke videoformate…

Vedlegg G

SORT ytelsestest

```python
# pragma: no cover
# noqa: D104

"""Script for benchmarking and testing different algorithms.

This file is not needed for the tracing module to function, but it's
    kept around
for easy testing.
"""

import json
import os
from typing import Any, List

import matplotlib.patches as patches
import matplotlib.pyplot as plt
import numpy as np
import skimage.io as io
from pycocotools.coco import COCO
from sort import sort

from tracing import tracker


def count_SORT(tracks: List[Any]):
    """Count idividual objects from sort."""
    count_individ_sort = set()
    for track_id in tracks:
        for track_id2 in track_id:
            count_individ_sort.add(track_id2[-1])

    return len(count_individ_sort)


def show_image(
    path: str,
    track_bbs_ids: List[Any],
    ax1: Any,
```

```
    colours: Any,
    fig: Any,
    idx: int,
):
    """For displaying images with boundingboxes."""
    I = io.imread(path)  # type: ignore
    ax1.imshow(I)

    for d in track_bbs_ids[-1]:
        d = d.astype(np.int32)
        ax1.add_patch(
            patches.Rectangle(
                (d[0], d[1]),
                d[2] - d[0],
                d[3] - d[1],
                fill=False,
                lw=1,
                ec=colours[d[4] % 32, :],
                label=d[4],
            ),
        )
    plt.savefig("video/frame-{:06d}".format(idx))  # type: ignore
    fig.canvas.flush_events()
    ax1.cla()


def main():
    """Entrypoint."""

    sort_tracker = tracker.Tracker(sort.Sort(1, 1))

    json_path = "instances_default.json"

    with open(json_path) as file:
        annotations_json = json.load(file)
    individ_json = set()
    for annon in annotations_json["annotations"]:
        if annon["image_id"] > 484:
            break
        try:
            individ_json.add(annon["attributes"]["track_id"])
        except:
            pass

    objects_in_json = len(individ_json)

    all_tracked_objects_in_json = len(
        [
            all_ann
            for all_ann in annotations_json["annotations"]
            if "track_id" in all_ann["attributes"]
            and all_ann["image_id"] <= 484
        ]
    )
```

```
92
93      coco = COCO(json_path)
94
95      coco = COCO(json_path)
96      catIds = coco.getCatIds(catNms=["Ørekyt"])  # type: ignore
97      imgIds = coco.getImgIds(catIds=catIds)  # type: ignore
98
99      # Create a "video". Also adds the image id so it's possible to find
         it for
100     # display later
101     frames = [(coco.loadAnns(coco.getAnnIds(imgIds=imgid)), imgid) for
        imgid in imgIds]  # type: ignore
102
103     for idx, frame in enumerate(frames[0:484]):  # type: ignore
104
105         bbx = []
106         local_detect = []
107
108         for ann in frame[0]:  # type: ignore
109             if (
110                 not ann["attributes"]["occluded"]
111                 and "track_id" in ann["attributes"]
112             ):
113                 box = tracker.BBox.from_xywh(ann["bbox"])  # type:
        ignore
114                 detect = tracker.Detection(box, ann["category_id"], 1,
        idx, ann["attributes"]["track_id"])  # type: ignore
115                 local_detect.append(detect)
116                 bbx.append(detect.to_SORT())
117
118         local_tracks = sort_tracker.update(local_detect)
119
120     miss_match = 0
121     for obj in sort_tracker.get_objects().values():
122         truths = set()
123         for o in obj.detections:
124             truths.add(o.true_track_id)
125         miss_match += len(truths) - 1
126
127     print("JSON:")
128     print(objects_in_json)  # type: ignore
129
130     print("detections:")
131     print(len(sort_tracker.get_objects()))  # type: ignore
132
133     MOTA = 1 - (
134         (
135             sort_tracker.get_misses()
136             + sort_tracker.get_false_positive()
137             + miss_match
138         )
139         / all_tracked_objects_in_json
140     )
141
```

```
    print("MOTA:")
    print(MOTA)

    distance = []
    ious = []
    for h, o in sort_tracker.get_matched():
        h_x = h[0] + ((h[2] - h[0]) / 2.0)
        h_y = h[1] + ((h[3] - h[1]) / 2.0)

        o_x = o[0] + ((o[2] - o[0]) / 2.0)
        o_y = o[1] + ((o[3] - o[1]) / 2.0)

        d_x = abs(h_x - o_x)
        d_y = abs(h_y - o_y)

        d = np.sqrt(d_x ** 2 + d_y ** 2)

        xi1, yi1 = max(h[0], o[0]), max(h[1], o[1])
        xi2, yi2 = min(h[2], o[2]), min(h[3], o[3])

        inter = abs(xi2 - xi1) * abs(yi2 - yi1)
        union = (
            abs(h[2] - h[0]) * abs(h[1] - h[3])
            + abs(o[2] - o[0]) * abs(o[1] - o[3])
        ) - inter
        ious.append(inter / union)

        distance.append(d)

    MOTP = sum(distance) / len(sort_tracker.get_matched())
    MOTP2 = sum(ious) / len(ious)
    print("MOTP:")
    print(MOTP)
    print(MOTP2)


if __name__ == "__main__":
    main()
```

Kodeliste G.1: SORT ytelsestest

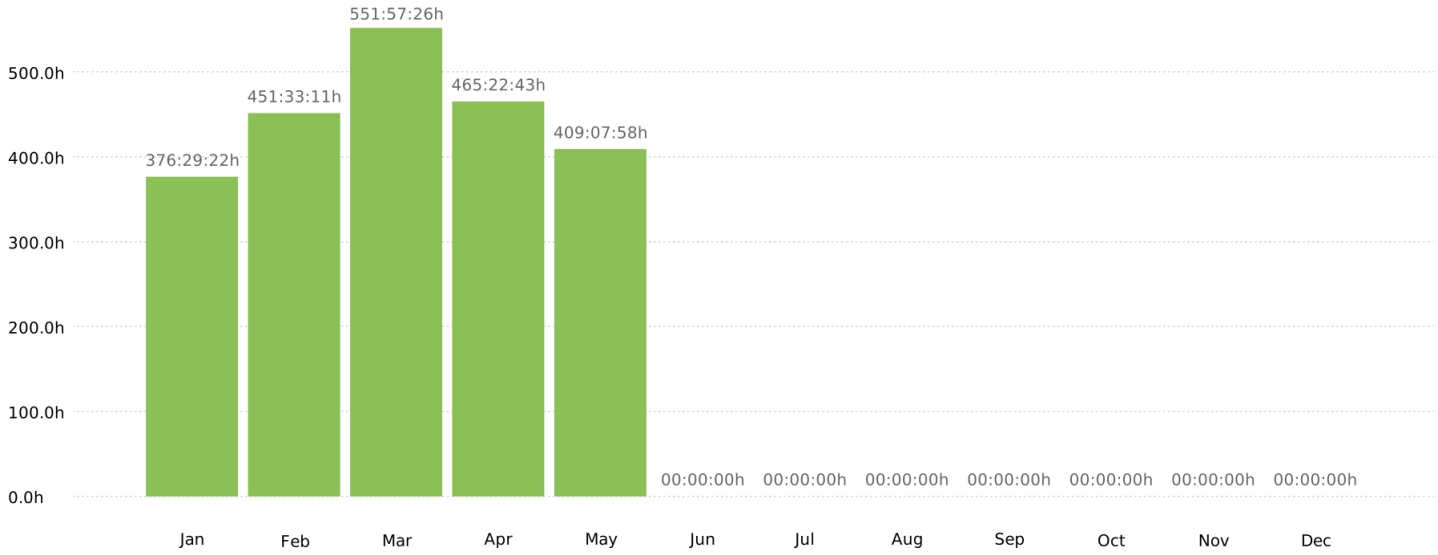
Vedlegg H

Timeføring

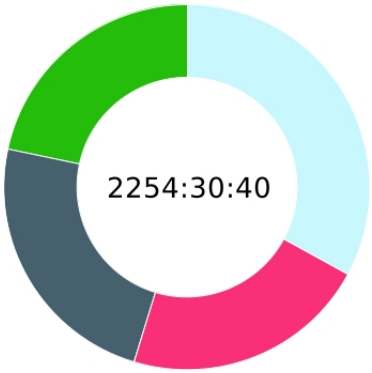
Summary report

01/01/2021 - 12/31/2021

Total: 2254:30:40

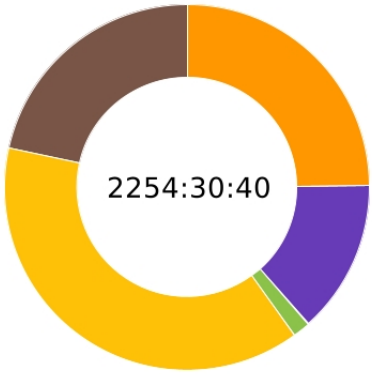


User



Birger J. Nordølum	487:57:17	21.64%
Eiriklavik	534:40:58	23.72%
Kristian	490:25:10	21.75%
TomR	741:27:15	32.89%

Project



Bachelor 2021 - NINA	485:30:39	21.54%
Code - NINA	868:14:39	38.51%
Infrastructure - NINA	33:29:00	1.49%
Infrastructure-old - NINA	01:44:41	0.08%
Meta - NINA	305:11:16	13.54%

●	Reports - NINA	560:20:25	24.85%

User / Project	Duration
Birger J. Nordølum	**487:57:17**
Bachelor 2021 - NINA	76:54:26
Code - NINA	208:18:37
Infrastructure - NINA	21:04:44
Meta - NINA	35:41:03
Reports - NINA	145:58:27
Eiriklavik	**534:40:58**
Bachelor 2021 - NINA	85:01:42
Code - NINA	251:38:05
Infrastructure - NINA	11:28:26
Infrastructure-old - NINA	01:44:41
Meta - NINA	37:45:03
Reports - NINA	147:03:01
Kristian	**490:25:10**
Bachelor 2021 - NINA	231:42:12
Code - NINA	172:26:37
Meta - NINA	30:50:40
Reports - NINA	55:25:41
TomR	**741:27:15**
Bachelor 2021 - NINA	91:52:19
Code - NINA	235:51:20

Infrastructure - NINA	00:55:50
Meta - NINA	200:54:30
Reports - NINA	211:53:16

Vedlegg I

Ytelsestest av BytesIO v. OpenCV

```python
#! /usr/bin/env python3

import os
import time

import torch
import multiprocessing

from core.model import Video
from core.services import VideoLoader
import cv2 as cv
import io
from PIL import Image


def bytesio(img):
    byte_io = io.BytesIO()
    Image.fromarray(img).save(byte_io, "png")
    byte_io.seek(0)
    return byte_io


def opencv(img):
    img = cv.cvtColor(img, cv.COLOR_BGR2RGB)  # type: ignore
    img_byte = cv.imencode(".png", img)  # type: ignore
    return io.BytesIO(img_byte)


video = Video.from_path(
    "/home/cuda/Downloads/File1 -[2020-07-18_08-17-54]-001.mp4"
)

batch_size = 100
video_loader = VideoLoader([video], batch_size)
chunk = batch_size // os.cpu_count()  # type: ignore

byte_frames = list()
start = time.monotonic()
```

```
with multiprocessing.Pool(os.cpu_count()) as pool:
    for frames, timestamp, _, _ in video_loader:
        byte_frames = pool.map(opencv, frames, chunk)
print(f"Multicore Opencv took  {time.monotonic()-start}")

byte_frames = list()
start = time.monotonic()
with multiprocessing.Pool(os.cpu_count()) as pool:
    for frames, timestamp, _, _ in video_loader:
        byte_frames = pool.map(bytesio, frames, chunk)
print(f"Multicore Bytesio took {time.monotonic()-start}")

byte_frames = list()
start = time.monotonic()
for frames, timestamp, _, _ in video_loader:
    byte_frames = [opencv(frame) for frame in frames]
print(f"Opencv took {time.monotonic()-start}")

byte_frames = list()
start = time.monotonic()
for frames, timestamp, _, _ in video_loader:
    byte_frames = [bytesio(frame) for frame in frames]
print(f"Bytesio took {time.monotonic()-start}")
```

Kodeliste I.1: Ytelsestest av ByteIO v. OpenCV

Vedlegg J

Ytelsestest av FFmpeg v. Opencv

```python
#!/usr/bin/env python

"""Benchmark comparing FFmpeg and OpenCV for extracing frames from
    video.

Times each iteration and appends delta into their respective list."""
import time

import matplotlib.pyplot as plt
import numpy as np

from video_ffmpeg import Video as FfmpegVideo
from video_opencv import Video as OpenCVVideo

vid1_path = "abbor-[2020-03-28_12-30-10]-000.mp4"
ffmpeg_video = FfmpegVideo.from_path(vid1_path)
opencv_video = OpenCVVideo.from_path(vid1_path)

frames = 10

# Measure FFmpeg iteration time
ffmpeg_delta = []
it = iter(ffmpeg_video)
for _ in range(frames):
    t0 = time.monotonic()
    _ = next(it)
    ffmpeg_delta.append(time.monotonic() - t0)


# Measure OpenCV iteration time
opencv_delta = []
it = iter(opencv_video)
for _ in range(frames):
    t0 = time.monotonic()
    _ = next(it)
    opencv_delta.append(time.monotonic() - t0)

### Benchmark ends here. ###
```

```
# The following is processing and displaying results.

# Output CSV formatted results.
dec = 3
print("metode,total,gjennomsnitt,median,standardavvik")
print(
    f"OpenCV,{round(sum(opencv_delta),dec)},"
    + f"{round(np.array(opencv_delta).mean(),dec)},"
    + f"{round(np.median(np.array(opencv_delta)),dec)},"
    + f"{round(np.array(opencv_delta).std(),dec)}"
)
print(
    f"FFmpeg,{round(sum(ffmpeg_delta),dec)},"
    + f"{round(np.array(ffmpeg_delta).mean(),dec)},"
    + f"{round(np.median(np.array(ffmpeg_delta)),dec)},"
    + f"{round(np.array(ffmpeg_delta).std(),dec)}"
)


fig1, ax1 = plt.subplots()
fig2, ax2 = plt.subplots()
fig3, ax3 = plt.subplots()

# Generate lineplot comparing FFmpeg and OpenCV over time
ax1.plot(opencv_delta, label="OpenCV")
ax1.plot(ffmpeg_delta, label="FFmpeg")
ax1.set_xlabel("Bilde")
ax1.set_ylabel("Tid i sekund")
ax1.legend()

# Generate boxplot for FFmpeg
bp1 = ax2.boxplot(
    ffmpeg_delta,
    positions=[1],
    notch=True,
    widths=0.35,
    patch_artist=True,
    boxprops=dict(facecolor="C2"),
)
# Generate boxplot for OpenCV
bp2 = ax2.boxplot(
    opencv_delta,
    positions=[2],
    notch=True,
    widths=0.35,
    patch_artist=True,
    boxprops=dict(facecolor="C0"),
)
ax2.legend([bp1["boxes"][0], bp2["boxes"][0]], ["FFmpeg", "OpenCV"])

bp3 = ax3.boxplot(
    opencv_delta[1:],
    positions=[2],
    notch=True,
```

```
    widths=0.35,
    patch_artist=True,
    boxprops=dict(facecolor="C0"),
)


fig1.savefig("result/lineplot_ffmpeg_v_opencv.pdf")
fig2.savefig("result/boxplot_ffmpeg_v_opencv.pdf")
fig3.savefig("result/cv_boxplot_ffmpeg_v_opencv.pdf")
```

Kodeliste J.1: Ytelsestest av FFmpeg v. Opencv

Vedlegg K

Video klasse med FFmpeg

,

```
class Video

        {...}

    def __iter__(self):
        """Class iterator."""
        self._current_frame = 0
        return self

    def __next__(self) → np.ndarray:
        """Get next item from iterator.

        Return
        ------
        np.ndarray
            One frame of video as `ndarray`.
        """
        if self._current_frame < self.frames:
            result = self.__get__(self._current_frame)
            self._current_frame += 1
            return result
        else:
            raise StopIteration

    def __get__(self, key) → np.ndarray:
        """Get one frame of video.

        Used by `__getitem__` when only one key is given.

        Returns
        -------
        numpy.ndarray
            One frame of video as `ndarray`.
        """
        if key < 0:
            raise IndexError

```

```
        if key >= self.frames:
            raise IndexError

        # ffmpeg filter docs:
        # http://ffmpeg.org/ffmpeg-filters.html#select_002c-aselect
        frame, _ = (
            ffmpeg.input(self._path)
            .filter("select", "eq(n, {})".format(key))
            .filter(
                "scale",
                self.output_width,
                self.output_height,
                -1,
            )
            .output("pipe:", vframes=1, format="rawvideo", pix_fmt="
    rgb24")
            .run(quiet=True)
        )
        return np.frombuffer(frame, np.uint8).reshape(
            [self.output_height, self.output_width, 3]
        )
        {...}
```

Vedlegg L

Video klasse med OpenCV

```
class Video:

        {...}

    def _scale_convert(self, img: np.ndarray) -> np.ndarray:
        """Convert and scale image using OpenCV.

        Converts image from BGR to RGB, and scales down to `self.output_{height,width}`

        Parameter
        ---------
        img : np.ndarray
            image to convert and scale

        Return
        ------
        ndarray:
            Scaled and converted image
        """
        new_img = cv.cvtColor(img, cv.COLOR_BGR2RGB)  # type: ignore

        new_img = cv.resize(  # type: ignore
            new_img,
            (self.output_width, self.output_height),
            interpolation=cv.INTER_AREA,  # type: ignore
        )
        return new_img

    def __iter__(self):
        """Class iterator.

        This never releases the VideoCapture. Not sure if it's kept alive, and
        if that's the case, this could cause a memory leak. To make sure this
        gets released, run `self.vidcap_release()`.

```

```
36          See Also
37          --------
38          Video.vidcap_release()
39
40          """
41          self._video_capture = cv.VideoCapture(self._path)  # type:
    ignore
42          self._video_capture.set(cv.CAP_PROP_POS_MSEC, 0)  # type:
    ignore
43          return self
44
45      def __next__(self) → np.ndarray:
46          """Get next item from iterator.
47
48          Return
49          ------
50          np.ndarray
51              One frame of video as `ndarray`.
52
53          """
54          err, img = self._video_capture.read()
55          if not err:
56              self.vidcap_release()
57              raise StopIteration
58          return self._scale_convert(img)
59
60      def __get__(self, key) → np.ndarray:
61          """Get one frame of video.
62
63          Used by `__getitem__` when only one key is given.
64
65          Returns
66          -------
67          numpy.ndarray
68              One frame of video as `ndarray`.
69
70          Raise
71          -----
72          RuntimeError :
73              if OpenCV fails to either read or set properties.
74          """
75          if key < 0:
76              raise IndexError
77
78          if key >= self.frames:
79              raise IndexError
80
81          self._video_capture = cv.VideoCapture(self._path)  # type:
    ignore
82          retval = self._video_capture.set(cv.CAP_PROP_POS_FRAMES, key)
    # type: ignore
83
84          if not retval:
85              raise RuntimeError(  # pragma: no cover
```

```
86                  f"Unexpected␣error␣when␣setting␣catpure␣property,␣{
    retval}"
87              )
88
89          retval, img = self._video_capture.read()
90
91          if not retval:
92              raise RuntimeError(
93                  f"Unexpected␣error␣when␣reading␣frame␣at␣{key}"
94              )  # pragma: no cover
95
96          self._video_capture.release()
97
98          return self._scale_convert(img)
99
100         {...}
```

Vedlegg M

SORT linking av rammer

```
class BBox:
# {...}
        def __eq__(self, o: BBox) -> bool:
                """Check if the two boundingboxes are witin 1 percent
    of eachother.

                Parameters
                ---------
                o : BBox
                Other BBox

                Return
                ------
                bool :
                        If they're equal
                """
                x1 = abs(self.x1 - o.x1)
                y1 = abs(self.y1 - o.y1)
                x2 = abs(self.x2 - o.x2)
                y2 = abs(self.y2 - o.y2)

                tol = 0.01
                return (
                        x1 < tol * abs(o.x1)
                        and x2 < tol * abs(o.x2)
                        and y1 < tol * abs(o.y1)
                        and y2 < tol * abs(o.y2)
                        )


class tracker:

# {...}

        def _connect_bb(
                self,
                tracked: np.ndarray,
                detect: List[Detection]
```

```
        ) → None:
                """Re-associate boundingboxes with a detection to
    determine the label.

                Parameters
                ----------
                tracked : np.ndarray
                        Tracked objects from self.tracker
                detect : List[Detection]
                        The detections to associate the boundboxes too
                """
                for t in tracked:
                        t_box = BBox(*t[0:4])
                        for d in detect:
                                if t_box == d.bbox:
                                        self._update_object(
                                                int(t[4]),
                                                Detection(
                                                d.bbox,
                                                d.label,
                                                d.score,
                                                d.frame,
                                                d.true_track_id
                                                ),
                                        )
```

Kodeliste M.1: SORT linking av rammer

Vedlegg N

Pipeline Definisjon

```
# ci/child-pipeline.yml

default:
  tags:
    - bachelor
  image: registry.gitlab.com/hitchhikers/container-images/python-
    quality:latest

stages:
  - prep
  - style
  - quality
  - test

variables:
  PIP_CACHE_DIR: $CI_PROJECT_DIR/.cache/pip
  POETRY_CACHE_DIR: $CI_PROJECT_DIR/.cache/poetry
  POETRY_VIRTUALENVS_CREATE: "false"
  POETRY_VIRTUALENVS_IN_PROJECT: "false"
  POETRY_VIRTUALENVS_PATH: $CI_PROJECT_DIR/.cache/poetry/venv

.cache_settings:
  cache:
    key: $CI_COMMIT_REF_SLUG-$PIPELINE_ROOT
    paths:
      - .cache/pip
      - .cache/poetry
      - $PIPELINE_ROOT/poetry.lock

build_cache:
  stage: prep
  extends: .cache_settings
  before_script:
    - cd $PIPELINE_ROOT
  script:
    - poetry install
  variables:
    GIT_DEPTH: "3"
```

```
style:
  stage: style
  before_script:
    - cd $PIPELINE_ROOT
  script:
    - isort --check --diff .
    - black --check --diff .
  cache: {}

docs:
  stage: style
  before_script:
    - cd $PIPELINE_ROOT
  script:
    - pydocstyle
  allow_failure: true
  cache: {}

pyright:
  stage: quality
  extends: .cache_settings
  before_script:
    - cd $PIPELINE_ROOT
    - poetry install
  script:
    - poetry run pyright .
  cache:
    policy: pull

tests:
  stage: test
  extends: .cache_settings
  before_script:
    - cd $PIPELINE_ROOT
    - poetry install
  script:
    - poetry run pytest --cov --cov-report xml
    - poetry run coverage report
  cache:
    policy: pull
  artifacts:
    reports:
      cobertura: $PIPELINE_ROOT/coverage.xml
```

Kodeliste N.1: Oppsett av en child-pipeline i GitLab.

Vedlegg O

Datasett verifiserings verktøy

```
"""Display images and annotations of a yolo dataset."""
import argparse
from typing import Any
from pathlib import Path
import skimage.io as io
import matplotlib
import matplotlib.patches as patches
import matplotlib.pyplot as plt
import numpy as np
matplotlib.use('tkagg')


def norm_to_abs(bb: np.ndarray, width: int, height: int) -> np.ndarray:
    """Convert normalized bounding box to pixel values

    Parameters
    ----------
    bb : numpy.ndarray
        Bounding box normalized, in format
        [class x_center y_center width height]
    width : int
        Image width.
    height : int
        Image height.
    """
    bb[1] *= width  # x
    bb[3] *= width  # width
    bb[2] *= height  # y
    bb[4] *= height  # height
    return bb


def show_image(
    path: str,
    bb: np.ndarray,
    ax1: Any,
    colours: Any,
    fig: Any,
```

```
) → None:
    """Display image with boundingboxes."""
    I = io.imread(path)  # type: ignore
    ax1.imshow(I)

    img_width, img_height = I.shape[1], I.shape[0]

    bb = [norm_to_abs(one, img_width, img_height) for one in bb]

    for c, x, y, w, h in bb:
        ax1.add_patch(
            patches.Rectangle(
                (x-(w/2), y-(h/2)),
                w,
                h,
                fill=False,
                lw=1,
                ec=colours[int(c)],
                label=c,
            ),
        )
    fig.canvas.flush_events()
    fig.show()
    ax1.cla()


def main() → int:
    parser = argparse.ArgumentParser()
    parser.add_argument("--path", type=Path)
    parser.add_argument("--step", default=1, type=int)
    args = parser.parse_args()

    # find all images in path
    images = sorted(list(args.path.glob("*.jpg")), key= lambda i: i.
    name)
    # filter acc. to step size
    images_subset = images[::args.step]

    plt.ioff()  # type: ignore
    fig = plt.figure()  # type: ignore
    ax1 = fig.add_subplot(111, aspect="equal")  # type: ignore
    colours = np.random.rand(10, 3)

    for img in images_subset:
        # generate filename for annotation file. Should have same
        # name as img.
        ann = img.name.split(".")[0] + ".txt"
        print(f"{img.name} and {ann}")
        anno_file = args.path / ann

        # read annotations from .txt file and convert to numpy array.
        with open(anno_file, "r") as f:
            # each line: class x_center y_center width height
            a = [one.split() for one in f.read().strip().splitlines()]
```

```
            bb = np.array(a, dtype=np.float32)
            assert bb.shape[1] == 5

        show_image(img, bb, ax1, colours, fig)

    return 0


if __name__ == "__main__":
    main()
```

Kodeliste O.1: CLI-verktøy for verifisering av datasett.

Vedlegg P

Møtereferater

Det ble avholdt nitten veiledningsmøter med veileder og syv iterasjonsmøter med NINA, se eksempel på referat fra disse i følgende sider.

Møtereferat

Sted: Digitalt

Dato / tid: 16.03.21 09:00
Emne: Veiledning
Deltagere: Birger, Kristian, Eirik, Tom-Ruben og Marius
Referent: Kristian

Agenda:

1. Rapportskriving
2. Minimumsløsning
3. Datasett
4. Brukertest NINA

Oppsummering:

Sak 1: Rapportskriving

Skriv noe, og heller gå tilbake og se på det senere om noen uker. Hver paragraf bør ha ett "stikkord", dersom ikke så hender det an en blander mye i samme paragraf. Om en klarer å lese stikkordene gjennom rapporten er det et tegn på at den er lettere å forstå. Kan også skrive stikkordene først, men krever mer planlegging og kunnskap om kva som skal stå i rapporten.

Må ikke nødvendigvis følge IMRAD strukturen, men må ha en logisk struktur på forklaringen av hvordan programmet fungerer.

Kor mye teori skal vi forklare, tom viser til tidligere rapporter som forklarer mye teknologier. Spørs litt på hvem en skriver for, skal kunne lese rapporten uten å ha lest altfor mye teori fra før. F.eks. lese rapporten uten faget AI. Kan forvente at de som leser er en medstudent med basisk programmering og kanskje OS faget. Fag som maskinlæring, tracing og API/Programstruktur burde forklares.

Kan skrive kort og konsist på 30 sider, men en kan også forklare mye på 50 sider. Spørs litt på "balansen" gjennom rapporten.

Refleksjon, hva fungerte og hva fungerte ikke. Skal levere refleksjonsnotat på slutten av bachelor separat. Trenger ikke mye refleksjon i rapporten på grunn av dette. Diskusjon gjerne mer vitenskapelig diskusjon på hva vi har fått til. Greit å få med endringer vi har gjort underveis i rapporten for å oppnå målene og hva som har fungert gruppemessig.

Mer fornuftig med mindre diskusjon del for del, og heller ha en større diskusjon på slutten der en ser på sammenkoblingen mellom delene.

Fordeler og ulemper med å spesialisere oss på områder. Har denne strategien påvirket sluttresultatet? Har dette gjort at vi har fått gjort mer?

Kan følge rapporten til Frode, men burde ha med alle overskriftene som kreves i bachelor. Rekkefølgen på overskriftene kan endres på om det gir mening for leseren. Få med bilder for å bryte litt tekst, dette er tungt å lese.

Sak 2: Minimumsløsning

Ser ut til at vi blir ferdig med minimumsløsning til planlagt dato 23.04.21. Ligger greit an i forhold til planen vi har satt opp. Gjenstår omtrent 1 uke med jobbing.

Sak 3: Datasett

Ser ut til at vi har problemer med datasett, spesielt gjedde. Tom viser. Marius sier at dersom det er enkeltbilder som feil detekteres på bakgrunn, kan vi lage logikk som ignorerer disse. Marius mener vi burde gå tilbake til cvat og sjekke at alt stemmer. Burde ha mer forskjellig bakgrunn dersom dette skal brukes flere plasseringer. Slik som det er nå kan bakgrunnen påvirke resultatet om vi henter ut data fra samme sted. Bruk av YouTube av undervannsvideo i andre settinger er anbefalt.

Sak 4: Brukertest NINA

Hva en ønsker å få ut av det. Kvalitativ eller kvantitativ data, hvordan testen er gjort. Kan være smart å lese litt rundt dette før vi planlegger testen. Kvalitetssikre prosessen slik det er etterprøvbart. Burde også tenke på hvilke type spørsmål/tilbakemelding vi ønsker å få besvart. Burde være obs på å ordlegge spørsmål på en nøytral måte slik det ikke påvirker resultatet. Burde også teste på noen "uvitende" først.

Til neste møte:

1. Få ned tekst i rapporten, selv om det er et grovt utkast.
2. Tenke på spørsmål vi ønsker å stille NINA under brukertest

Møtereferat

Sted: Digitalt

Dato / tid: 22.04.21 12:00
Emne: Iterasjonsmøte med NINA
Deltagere: Birger, Kristian, Eirik, Tom-Ruben, Knut Marius og Tobias
Referent: Kristian

Agenda:

1. Ny funksjonalitet
2. Demo av applikasjon
3. Bruk av statistikk
4. Potensial i sky

Oppsummering:

Sak 1: Ny funksjonalitet

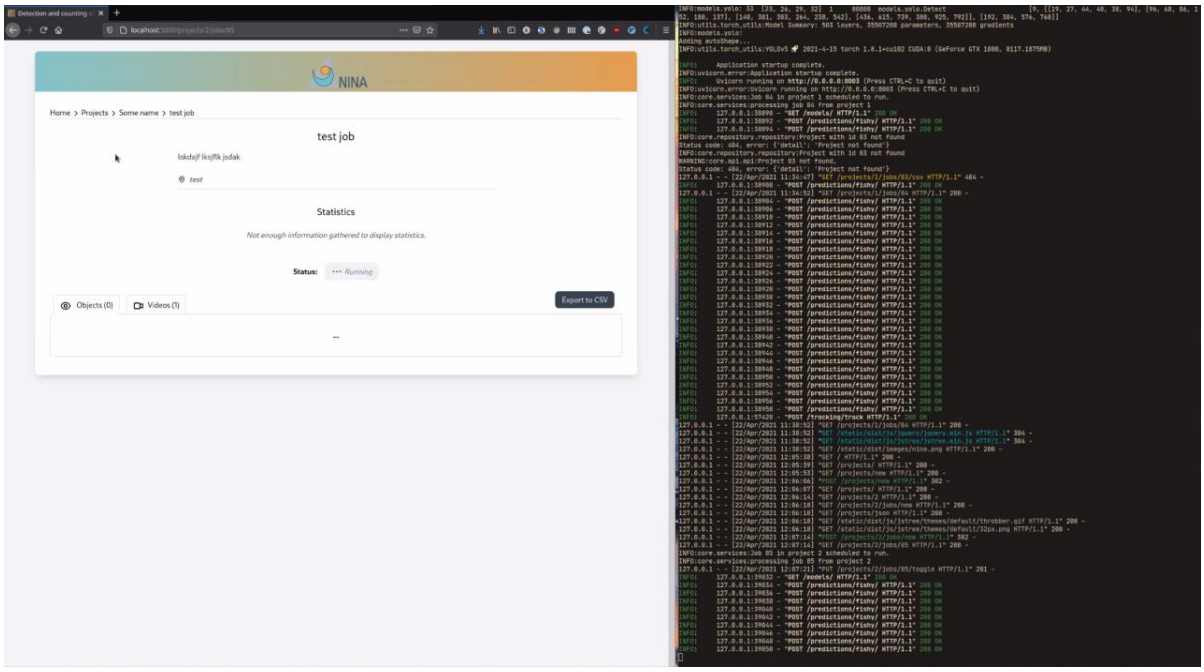
Lagring av prosessering underveis: Slipper å starte en jobb på nytt om en avslutter programmet midt i en kjørende jobb.

Datasett og trening: Fått i orden trening av maskingjenkjenning av fisk. Ut fra rask test ser resultatet relativt pålitelig ut.

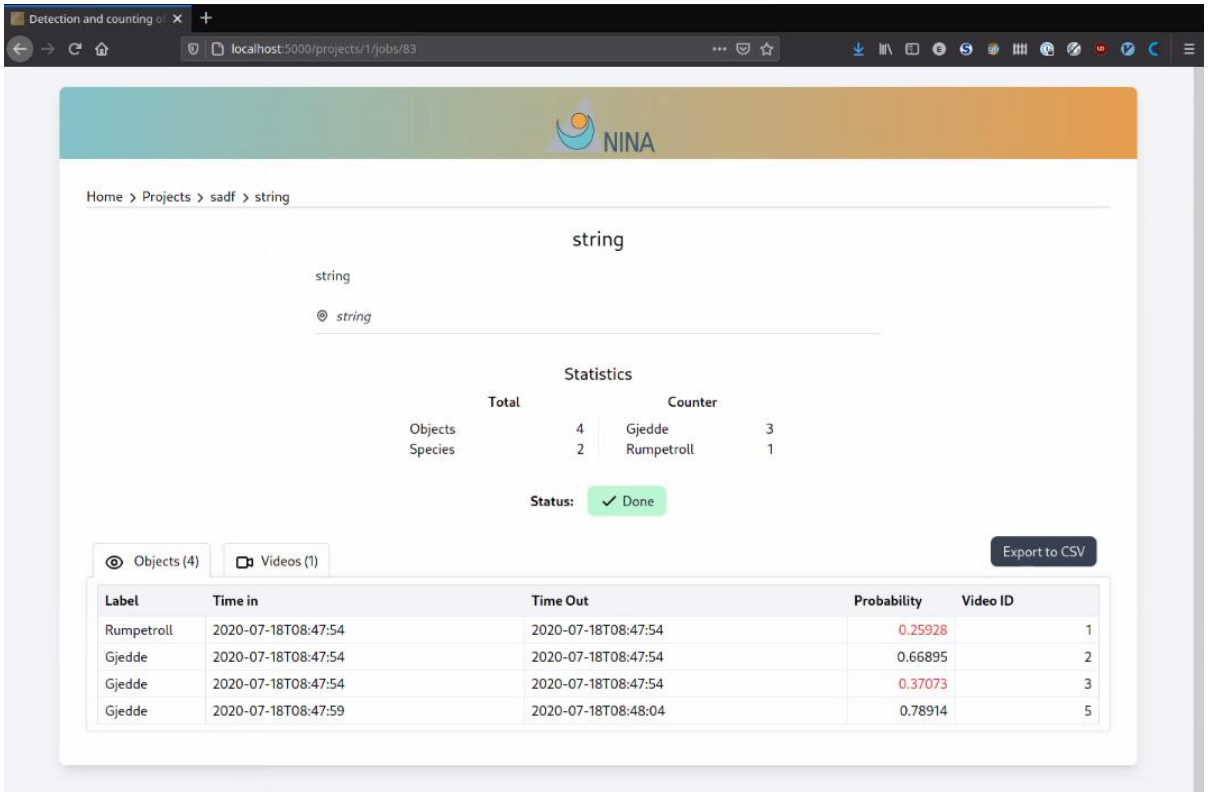
Brukergrensesnitt: Endret på småting som NINA nevnte under brukertest. For eksempel at en må trykke på start før programmet prosesserer en jobb.

Sak 2: Demo av applikasjon

Eirik viser demo av applikasjonen og hvordan den fungerer. Avbildet under er en kjørende jobb. Resultat nederst på høyre side viser at programmet jobber.

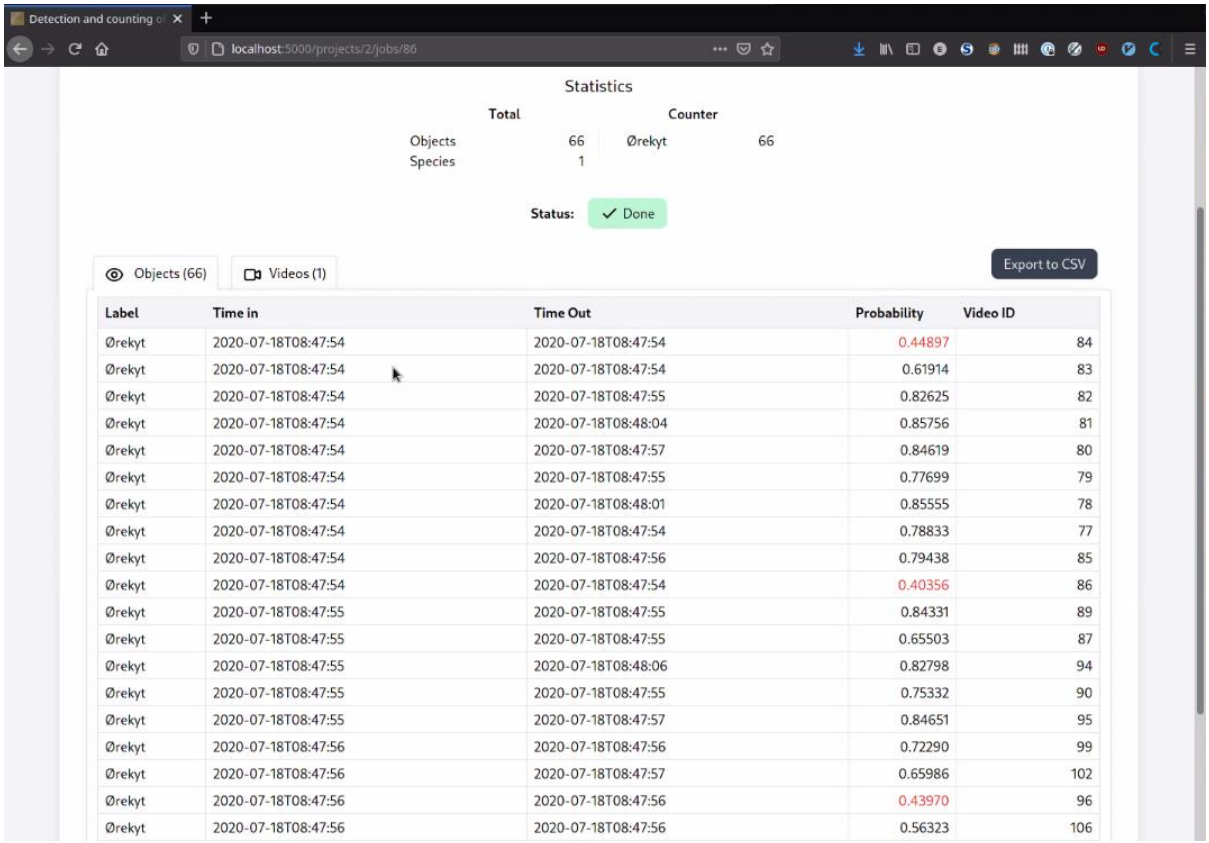


Resultatskjerm av en fullført jobb er avbildet under. Viser objekter som er detektert sammen med kalkulert sannsynlighet. Sannsynligheten viser gjennomsnittet av sannsynligheten i alle bilder fisken er oppdaget. Sannsynligheter under 0.5 blir markert som rød i brukergrensesnittet.

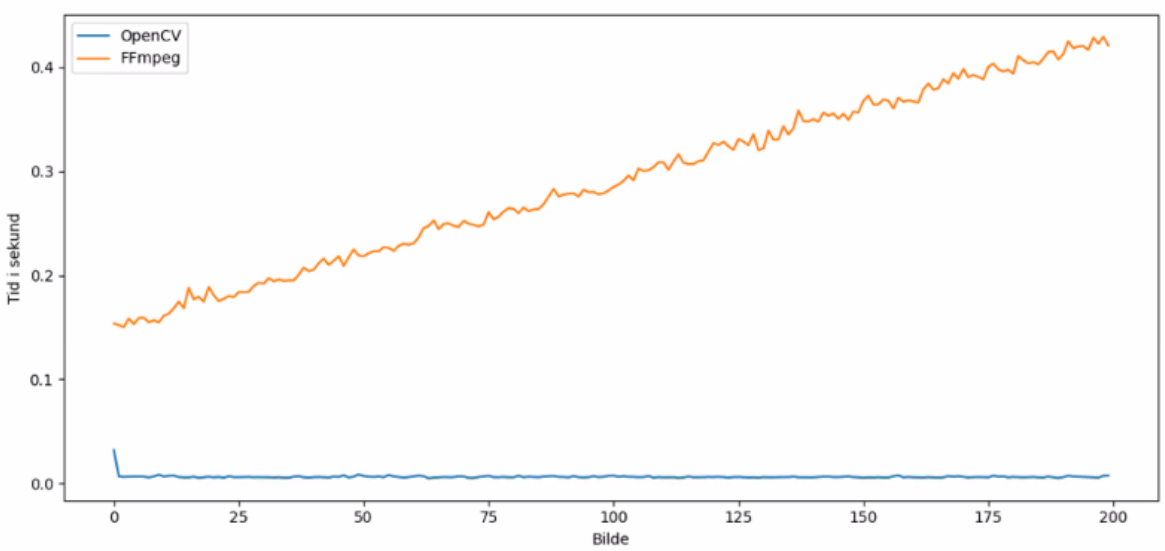


Resultat av video med mange ørekyt er avbildet under. Manuell opptelling gir omtrent 33 fisker. Ut fra programmet er det feil antall objekter som er funnet. Dette kan bli forbedret med å endre noen parametere på sporingen. Ved manuell korrigering av disse variablene har resultatet blitt redusert til 37 fisker.

Parameterne er noe vanskelig å justere siden de fungerer bra på noen videoer, og dårlig på andre.

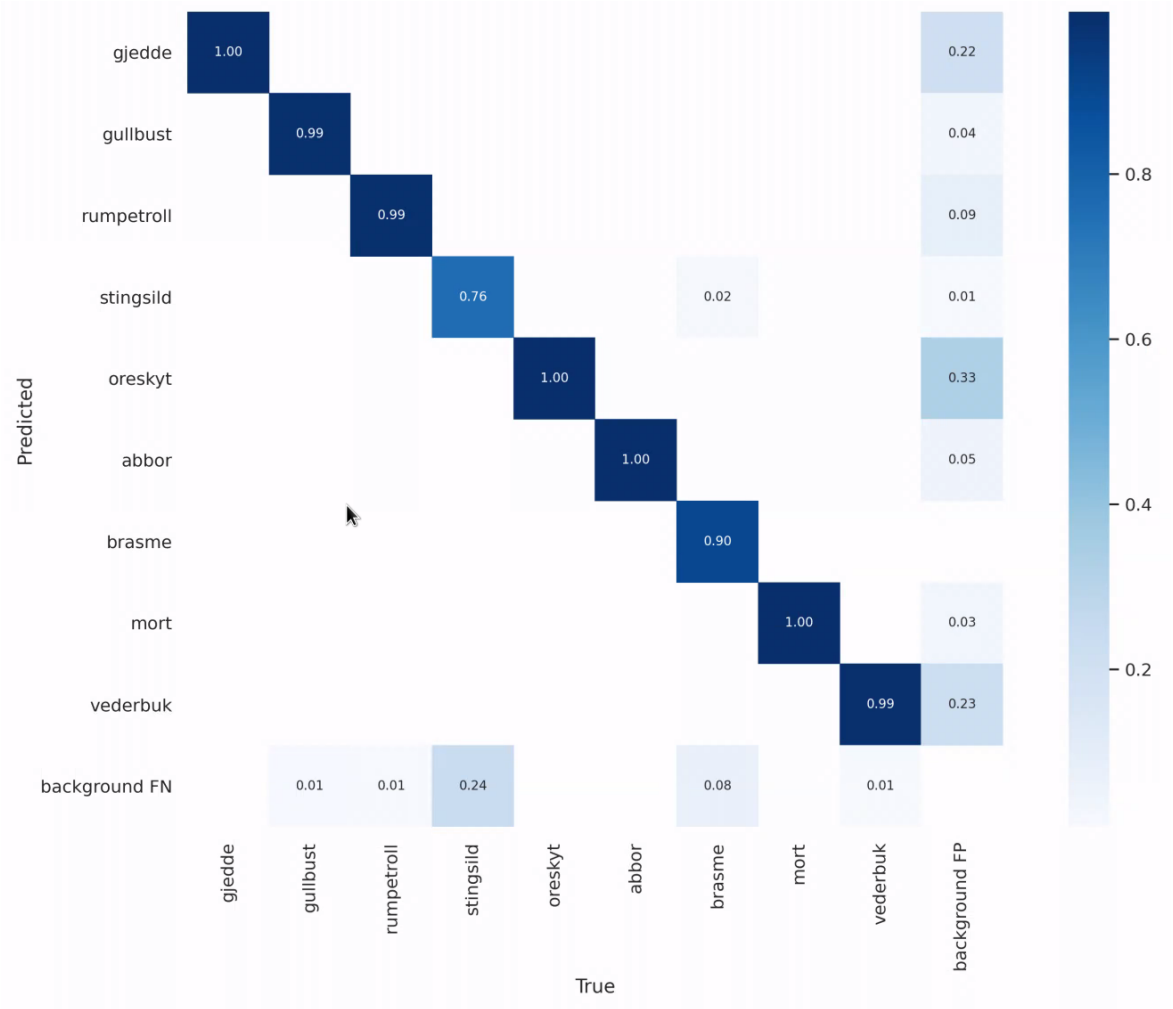


Ytelse er også noe vi har jobbet mye med. Før brukte vi programmet FFmpeg til å hente ut bilder i fra video. Grafen viser tiden det tar å hente ut bilder fra en ramme i sekunder i Y aksen, og indeks av bilde i video på X akse. FFmpeg bruker lengre tid jo lengre inn i videoen den er. Blå linje under viser tiden det tar med ny løsning OpenCV. Dette har å si på hvor lang tid programmet bruker på å laste inn videoer til prosessering.



Ytelsen til hele applikasjonen er basert på mange faktorer. Spesielt hastighet på disk, og rå datakraft i form av GPU og CPU. Under testing programmet, bruker det omtrent 9 sekunder per 3 sekunder med video. Dette var gjort med en PC med Ryzen 9 5900x, NVIDIA GTX2070 Super, med videofil lagret på SSD. Denne ytelsen vil ikke være representativ på en laptop pc, da disse datamaskin spesifikasjonene er mye kraftigere enn det en vanlig laptop har tilgjengelig.

Confusion matrix til den nylig trente modellen som blir brukt til deteksjon.



Sak 3: Bruk av statistikk

NINA forklarer at antall objekter er ikke alltid så nøye, er mer for å få et inntrykk av hvilke arter som er ute. Om en ønsker å se nøyere på data kan det være en god pekepinn på når det er fisk, som kan korrigeres ved å se på video. Det beste hadde vært at objektene var korrekte, men NINA forstår at dette er vanskelig å få til.

Plassering av kamera har også mye å si på hvilke fisker som blir sett på kamera. NINA forteller dersom de plasserer kamera nærme vegetasjon, vil mindre fisk som søker ly i vegetasjon bli fanget opp mer på kamera. Dette kan gjøre statistikken feil om målet for opptaket er hvor kameraet peker, i stedet for fisk som svømmer forbi helt inntil kamera.

Sak 4: Potensial i sky

Det finnes løsninger for å leie datakraft i skyen. Dette betyr at en kan leie datakraft til applikasjonen i stedet for å kjøpe dedikert maskinvare spesifikt til å kjøre applikasjonen. Her kan en leie nøyaktig etter hva man trenger i form av ytelse og budsjett.

Til neste møte:

1. Gruppen gir tips til pc på komplett som kan brukes til deteksjon.

Møtereferat

Sted: Digitalt

Dato / tid: 25.02.21 09:00
Emne: Iterasjonsmøte
Deltagere: Birger, Kristian, Eirik, Tom-Ruben
Referent: Kristian

Agenda:

1. Gå gjennom åpne issues
2. Velge issues til neste iterasjon

Oppsummering:

Sak 1: Issues

Fjerne store og retrieve job på repository nivå. Erstattet av update funksjon som tar inn et helt prosjekt som skal oppdateres. Jobber vil bli oppdatert dersom oppdatert project objekt har endret jobber.

Trening av deteksjonsmodell blir ferdig i dag.

Sak 2: Issues neste iterasjon

Core:

- Video class
- Frame class
- Detection class
- Object class
- Location string in Job

Tracking

- Tracking API

Detection:

- Create model

View:

- Create project
- Create job

Repository:

- Repository for video og frame
- Object & Detection (kombinert med prosjekt aggregat)

Artsgjenkjenning av fisk

NTNU

Kunnskap for en bedre verden